National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

NL-339 (r. 88/04) c

Canada

University of Alberta

Performance of Bus-Type Local Area Networks Under Correlated Traffic Patterns

by

Ambrose Shu Yee Chow

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Masters of Science

Department of Computing Science

Edmonton, Alberta
Fall 1991

National Library of Canada
Bibliothèque nationale du Canada

Canadian Theses Service
Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-69949-3

Canada

UNIVERSITY OF ALBERTA

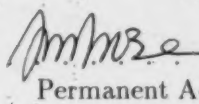
RELEASE FORM

NAME OF AUTHOR: Ambrose Shu Yee Chow
TITLE OF THESIS: Performance of Bus-Type Local Area Networks Under Correlated Traffic Patterns

DEGREE: Masters of Science
YEAR THIS DEGREE GRANTED: 1991

Permission is hereby granted to the UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis not extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

(Signed) 

Permanent Address:
137, Kiu Kiang Street,
8th floor, Kowloon,
Hong Kong.

Date: 16 September 1991

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled **Performance of Bus-Type Local Area Networks Under Correlated Traffic Patterns** submitted by **Ambrose Shu Yee Chow** in partial fulfillment of the requirements for the degree of Masters of Science.

Pawel Gburzynski. . . . . (Supervisor)

. . . . . .

Piotr Rudnicki. . . . . .

Werner Joerg. . . . . .

Ehab El-Mallah. . . . . .

. . . . . .

Date: 16 September 1991

Abstract

Performance of local area networks is investigated either by simulation or by mathematical modeling. In both cases, the traffic is generated according to some distribution, for example, uniform or exponential. The actual traffic of a local area network is very difficult to model by a single distribution. In most cases, traffic is correlated. In this thesis, four bus-type local area networks (Ethernet, Token-Bus, CSMA/CD-DP, and Expressnet) are selected, and their performance is studied under different kinds of traffic that occur in reality. Our study uses a simulation system, called LANSF. It is found that the performance of local area networks heavily depends on traffic patterns. Under uncorrelated traffic, Ethernet, the most common local area network, has poor performance when the traffic is heavy. However, under file server traffic, Ethernet shows good performance even at busy traffic. Other local area networks, Token-Bus and CSMA/CD-DP, also show different performance under different traffic patterns. Under file server traffic, in some cases, Token-Bus has worse performance than Ethernet. Also, in this thesis, the protocol of Token-Bus is modified: a station is allowed to hold the token when the number of messages waiting for transmission at the station reaches a predefined value. The performance of this modified Token-Bus is also studied. Most local area networks installed have

a mixture of different kinds of traffic. Two typical traffic patterns are RPC and file server traffic. The result of this thesis shows that Ethernet is overall a good local area network to use.

Preface

This thesis has seven chapters. Chapter One describes the objective of the thesis and explains why Ethernet, Token-Bus, Expressnet and CSMA/CD-DP are chosen for the investigation. It also has a section to describe discrete-event simulation under LANSF. Chapter Two describes the protocols of Ethernet, Token-Bus, Expressnet and CSMA/CD-DP. A brief survey of some recent results related to the performance of the above networks is also included. Chapter Three describes some characteristics of uncorrelated traffic, remote procedure call traffic and file server traffic. Chapters Four, Five and Six present simulation results and give explanations of the observed performance of each network under uncorrelated traffic, remote procedure call traffic, and file server traffic respectively. Finally, Chapter Seven summarizes many of the observations found in the simulation experiments. From these observations, conclusions are drawn. These conclusions state the suitability of using each of the four networks mentioned above given that the expected dominating traffic pattern is either uncorrelated traffic, remote procedure call traffic or file server traffic.

Acknowledgements

I have to thank Dr. Pawel Gburzynski, my supervisor, for his guidance and assistance during the entire period of this work. I also wish to thank the examination committee members, Professor Werner Joerg, Dr. Ehab El-Mallah, and Dr. Piotr Rudnicki. Their comments were valuable in improving the thesis. I thank Mr. Charles Chirchir for his help and suggestion to my thesis. I particularly thank Mrs. Edith Drummond for her support, her kindness and her encouragement during these years of my studying. Finally, I have to thank Miss Winslowe Lacesso for the proofreading of the manuscript. Without her this thesis will never be finished.

Contents

1 Introduction **1**

1.1 Objective 1

1.2 Choice of Local Area Networks for Investigation 2

1.3 Simulation Environment 3

1.4 Time and Distance Units 10

2 Protocols and Results of Previous Studies **11**

2.1 Introduction 11

2.1.1 Ethernet 12

2.1.2 Expressnet 15

2.1.3 Token-Bus 17

2.1.4 CSMA/CD-DP 20

2.1.5 General Network Parameters 22

2.2 Performance Measures 22

2.3 Previous Results on the Performance of Selected LANS 24

2.3.1 Ethernet 24

2.3.2 Token-Bus 26

2.3.3 Expressnet 27
2.3.4 CSMA/CD-DP 28
2.3.5 Summary 29

3 Traffic Patterns **31**

3.1 Uncorrelated Traffic 31
3.2 Remote Procedure Call Traffic 32
3.3 File Server 33
3.4 Network Load 36

4 Performance Under Uncorrelated Traffic **38**

4.1 An Uncorrelated Traffic Model 38
4.2 Simulation 39
4.3 Performance Comparison 39
4.3.1 Ethernet 39
4.3.2 Token-Bus 40
4.3.3 Token-Priority 42
4.3.4 CSMA/CD-DP 42
4.3.5 EXPRESSNET 43
4.4 Conclusion 45

5 Performance Under Remote Procedure Call Traffic **47**

5.1 Model 47
5.2 Simulation 48
5.3 Observations 49
5.3.1 Single Process 49

5.3.2 Three Process 50
5.4 Explanation of the Observation 50
5.5 Conclusion 51

6 Performance Under File Server Traffic **56**
6.1 A File Server Model 56
6.1.1 Software and Hardware Overhead 56
6.1.2 Request and Reply 57
6.1.3 Request Behaviour 57
6.2 Simulation 58
6.3 Observations and Explanation 60
6.3.1 Performance Comparison Among Various Networks 60
6.3.2 Performance with Different Numbers of Servers 61
6.3.3 Performance with Servers at Various Locations 62
6.3.4 Comparison between Token-Bus and Token-Priority 65
6.4 Conclusion 65

7 Summary **74**
7.1 Performance with Uncorrelated Traffic 74
7.2 Performance with RPC Traffic 76
7.3 Performance with File Server Traffic 76
7.4 Conclusion 78

A Programs of Protocols **83**
A.1 Ethernet 83
A.2 Token-Bus 86

A.3 CSMA/CD-DP 88
A.4 Expressnet 92

Chapter 1

Introduction

1.1 Objective

Network performance has been investigated by creating mathematical models or by simulation. In both cases, traffic is generated according to some statistical processes. In analytical modeling, the traffic is assumed usually to be generated by a Poisson process with a given mean value. In simulation, traffic is generated usually by assuming that message interarrival time and message length are distributed exponentially or uniformly.

Those kinds of traffic are called uncorrelated. In practice, real traffic is always correlated and is very difficult to describe by a mathematical model. For example, given two processes A and B residing in two different stations: A has sent a message to B. It is very likely that A will suspend all its work until it has received the response from B. Typical examples of such correlated traffic could be found in systems of remote procedure call and file server. Correlated traffic depends on the

system parameters, rather than some statistical distribution, e.g. the service time, the number of clients and servers in the system, the message length, and the number of processes in a station. In particular, the correlated traffic of a remote procedure call and a file server system depends on how long a station has to wait for a reply.

The main purpose of this research is to investigate the performance of local area networks under different kinds of traffic patterns. Three different kinds of traffic patterns are considered: the uncorrelated traffic pattern, the traffic pattern of a remote procedure call system and the traffic pattern of a file server system.

1.2 Choice of Local Area Networks for Investigation

Four local networks are selected for investigation. They are Ethernet [9], Token-Bus [14], CSMA/CD-DP [8], and Expressnet [20] and [4]. The performance of these protocols under uncorrelated traffic has been extensively studied. Ethernet has poor performance at busy and moderate traffic. CSMA/CD-DP is a collision-free network based on CSMA/CD. Token-Bus is another collision free bus type network. Expressnet is a fast network with good performance under heavy traffic. The protocol of Ethernet is the simplest one with the least amount of overhead to transmit a packet, and it is also the most common local area network. Token-Bus, another common local area network, and CSMA/CD-DP are especially designed for heavy traffic. But, under the correlated traffic, will Token-Bus and CSMA/CD-DP still perform better than Ethernet? Will Expressnet still have the best performance? This is the reason why the four networks are chosen for the investigation.

1.3 Simulation Environment

The simulation experiments have been implemented using a system called, LANSF [5]. LANSF is a discrete-time *event-driven* simulation environment, in which the simulation time elapses by the occurrence of events. Event-driven simulation can be used to simulate systems consisting of physical processes which are linked and activated by events as shown in Figure 1.1.

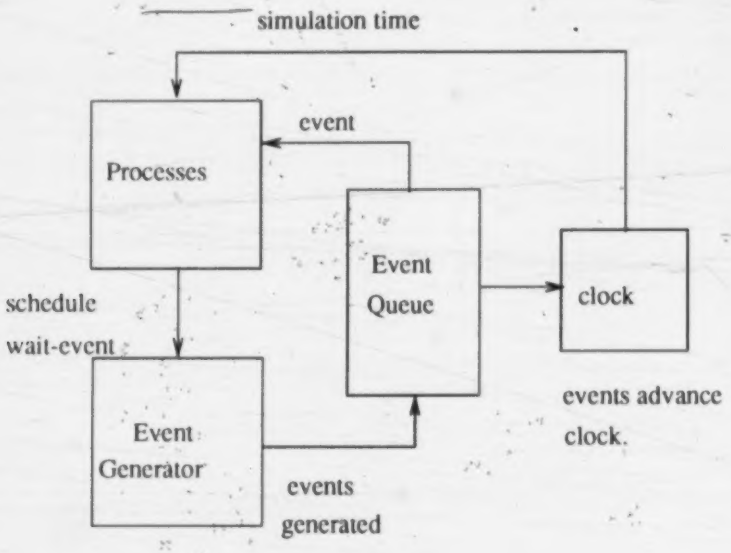


Figure 1.1 Structure of an event-driven simulator.

The processes issue the "wait-event", passing the parameters to the event-generator which generates the events and puts them into the event-queue. In each execution cycle, the earliest event is removed from the queue. This event then updates the global simulation time, and also wakes up the waiting process to perform the specified actions. After that, the process usually issues another event request,

and the same steps are repeated. A general format of a process looks like the following:

```
process () {
    do-the-action(action);
    schedule-next-event(action, event);
    goto-sleep(); }
```

A more detailed example of a car-wash process is now in order. In this example, there is a washing process, a waxing process and a car arrival process. The car arrival process generates the arrival of a car according to a statistical distribution and inserts the car into the car-wash queue. The washing process and the waxing process simulate the washing and the waxing. The amount of time needed to wash a car is computed from a statistical distribution. Similarly, the amount of time needed to wax a car is calculated from a second distribution. Initially, when the simulation starts, each process is wakened by a special event issued from the main program.

At first, the car arrival process calculates the car interarrival time. This car interarrival time is passed to the event generator as the parameter of the wait-event issued from the car arrival process. The car arrival process then goes to sleep and waits for the car-arrival-event. When the car arrival process receives the car-arrival-event it wakes up and generates a car. The car arrival process inserts the car into the wash-queue. After that, the car arrival computes the car interarrival time, issues the wait-event, goes to sleep and waits for another car-arrival-event. When the washing process is wakened up initially, it checks to see if the car-wash queue has cars waiting. If the car-wash queue is empty, it issues the wait-event to the

event generator, goes to sleep and waits for the wash-arrival-event. If there are cars waiting in the car-wash queue or the wash-arrival-event event arrives, the washing process takes the first car in the queue and calculates the amount of time according to the statistical distribution. This amount of time corresponds to the washing time. The washing process then issues the wait-event and passes the washing time as the parameter to the event generator. It then goes to sleep and waits for a wash-finish-event. When the washing process is wakened by the wash-finish-event the car has been washed. The washing process then inserts the car into the car-wax queue, and repeats the task again. Similar to the washing process, when the waxing process is wakened up initially, it checks to see if the car-wax queue is empty. If the car-wax queue is empty, the waxing queue issues the wait-event to the event generator, goes to sleep and waits for the event, wax-arrival-event. If the waxing process receives the event, wax-arrival-event or the car-wax queue is not empty, the waxing process picks up the first car in the queue. It then computes the waxing time according to the statistical distribution. This amount of time is passed as the parameter of the wait-event to the event generator, and simulates the amount of time needed to wax a car. The waxing process then goes to sleep and waits for the event, wax-finish-event. When the waxing process is wakened up by this event, the car waxing is finished. The waxing process sends the car to the gate, and then repeat the job again.

The simulation can be implemented by using a program loop. In each cycle, the event queue is sorted with the earliest event at the top. The earliest event is picked up and the event time is passed to the global clock to update the simulation time. The event is passed to the waiting process which is then wakened up to perform

the action and issue the next event request. If the time parameter of the wait-event issued from a process has a value, the event generator generates the event right away and puts it into the event queue. In some cases, the time parameter of a wait-event does not have a value yet, for example the car-arrival-event. The event-generator then puts such special wait-events onto a list, and checks with the wash-queue and the wax-queue in each cycle to generate the request event. The wait-event has three parameters as shown: wait-event(action, event, time). The action parameter indicates the action the process will perform when it is wakened up. The event parameter and the time parameter indicate the event that the process is waiting for and the time that the event will occur. The program structure of the simulation is shown in Figure 1.2.

The arrows indicate the direction of the program flow. The structure descriptions are:

- **car-wash()**: the process simulating the washing.
- **car-wax()**: the process simulating the waxing.
- **car-arrive()**: the process generating the car arrival.
- **event generator**: generates the events for which the processes are waiting.
- **event queue**: events are sorted with the earliest events at the top of the queue.
- **wash-queue**: the queue of cars waiting to wash.
- **wax-queue**: the queue of cars waiting to wax.

- **global time clock**: The clock keeps the simulation time which is updated by events.

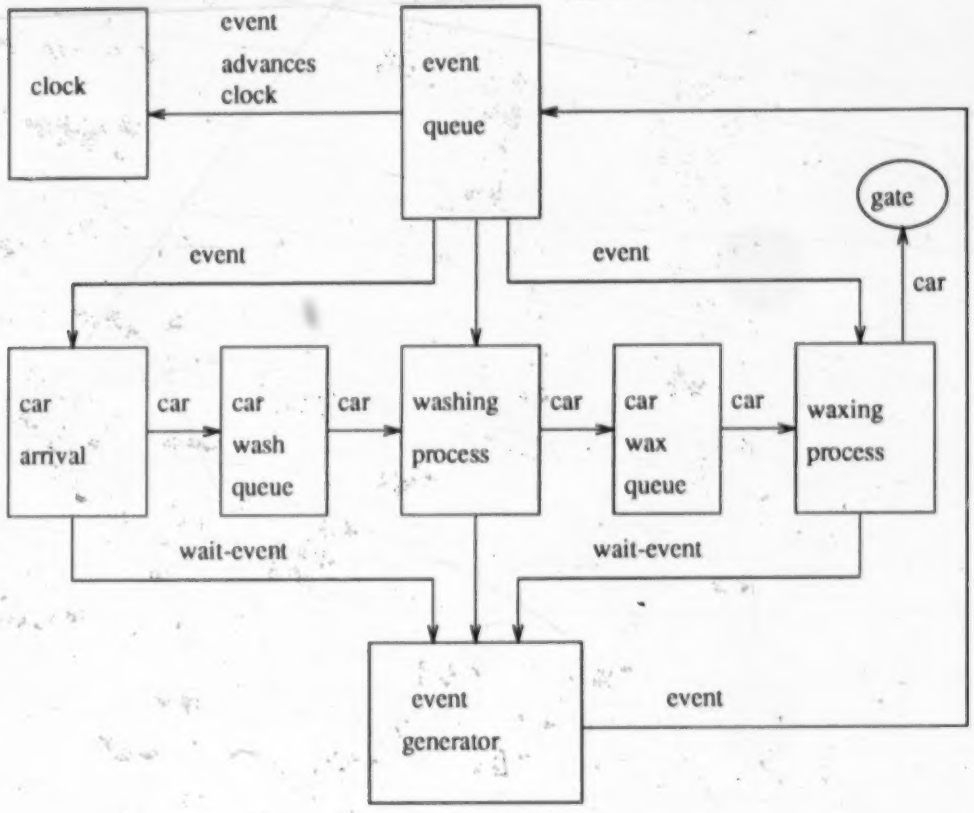


Figure 1.2 Car-Wash block structure.

The program structures of the processes are shown in the following:

```
car-wash () {
    case (action) :
        initialization:
            if wash-queue-empty then {
```

```
                wait-event(start-action-wash, wash-arrival-event, no-time-yet);
                goto sleep; }
            else {
                pick-a-car-wash(car);
                continue-at(start-action-wash); }
        start-action-wash:
            wash-time = statistical-distribution(average-wash-time);
            wait-event(finish-wash-action, wash-finish-event, wash-time);
            goto sleep;
        finish-wash-action:
            to-wax-queue(car);
            continue-at(initialization);
    }


car-wax () {
    case (action) :
        initialization:
            if wax-queue-empty then {
                wait-event(start-action-wax, wax-arrival-event, no-time-yet);
                goto sleep; }
            else {
                pick-a-car-wax(car);
                continue-at(start-action-wax); }
```

```
        start-action-wax:
            wax-time = statistical-distribution(average-wax-time);
            wait-event(finish-wax-action, wax-finish-event, wax-time);
            goto sleep;
        finish-wax-action:
            to-gate(car);
            continue-at(initialization);
    }

car-arrive () {
    case (action) :
        initialization:
            car-interarrival-time = statistical-distribution(average-interarrival);
            wait-event(generate-car-action, car-arrival-event,
                car-interarrival-time);
            goto sleep;
        generate-car-action:
            to-wash-queue(car);
            continue-at(initialization);
    }
```

LANSF is a system that provides functions and procedures to write processes for simulation of communication networks. For example, in the simulation of the

Ethernet protocol, there could be two processes at a station, the "transmitter()" and the "receiver()". The "transmitter()" waits for the arrival of packets, transmitting the packet upon the satisfaction of interpacket space, listening to the bus, aborting the transmission in case of collision, transmitting the packet again after the back-off, and ending a successful transmission. The "receiver()" is simpler, it just waits for the packet transmitted to it. The LANSF code of the Ethernet "transmitter()" and "receiver()" is outlined in Appendix A.

1.4 Time and Distance Units

Time can be measured in bits. If the transmission rate is R bits per second, then it takes $1/R$ seconds to insert a bit into the bus. Thus, the $1/R$ second is the one bit unit time. The signal propagation speed is about 0.7c (where c is the speed of light). If the transmission rate is known, distance can be expressed in bits. For example, if the port rate is 10 Mbs, then 1 bit can be transmitted in 10^{-7} sec, and this time interval corresponds to 21 meters. Through this report, time and distance are expressed in bits, unless it is specified otherwise. In LANSF we define an *indivisible time unit* (denoted **ITU**) to be the time required for a communication port to insert one bit into a bus link. For instance, if a port transmits data at the rate of 10 Mbits/sec then 1 ITU equals 10^{-7} sec.

Chapter 2

Protocols and Results of Previous Studies

2.1 Introduction

The IEEE 802 Committee [15] has developed a three layer architecture for local area networks. The top layer is the **logical link control**, the middle is the **medium access control** and the bottom is the **physical layer**. The logical link control layer provides one or more **service access points**. A service access point is a logical interface between two adjacent layers. The logical link control layer assembles data into framed packets with the address and the CRC field. It also disassembles packets, and performs address recognition and CRC validation. The medium access control layer manages the access to the communication link. The physical layer is responsible for the encoding and decoding of signals, preamble generation and removal, and bit transmission and reception. There are various kinds of medium

access control protocols, e.g. CSMA/CD, Token-Bus, and Token-Ring. The medium access control protocol is commonly referred to as simply the protocol of the local area network. The protocols of Ethernet, Expressnet, Token-Bus and CSMA/CD-DP are briefly described below.

2.1.1 Ethernet

Ethernet is perhaps the most widely known local area network. It uses the protocol of CSMA/CD, which is also known as "Listen while you talk". CSMA/CD is a refinement of CSMA. The main difference between these two protocols is that in CSMA a station does not detect a collision when it is transmitting a packet. So, under CSMA the channel remains unusable for the duration of the transmission of damaged packets. If the packet is long this waste can be considerable. In CSMA/CD the transmission will be aborted when a collision is detected. An interesting question is how long it takes a station to detect a collision. The answer can be obtained by considering the following case. Two stations A and B are separated by a distance of one propagation delay, τ, (i.e. they are at the ends of the link). A transmits a packet to B. Right at the moment when this packet reaches B, B starts to transmit a packet, which collides with the packet from A. Thus it takes 2τ for A to detect the collision. Also, it means that the packet should have the length of at least 2τ. Otherwise, if the above happens, A will think that the transmission is successful.

There are two major variations of CSMA/CD, based on the action a station will perform when it senses the link busy. One variation is the **nonpersistent** CSMA/CD, and the other is the **persistent** CSMA/CD. For nonpersistent CSMA/CD, if the channel is sensed busy, a station will wait for a certain amount

of time and then repeat to sense the channel again. In persistent CSMA/CD, if the channel is busy, a station continues to sense until the channel is found idle. Persistent CSMA/CD can also be classified into two classes, the 1-persistent and the p-persistent. For 1-persistent CSMA/CD, if the channel is found idle, a station will transmit a packet with probability 1. While in p-persistent, if the channel is idle, a station will transmit with probability p, and delay one time unit with probability $(1 - p)$. The time unit is typically equal to the maximum propagation delay. Also, CSMA/CD can be slotted in the sense that the time axis is divided into slots and each slot has the size of τ (normally the propagation delay). All stations are synchronized and start transmissions only at the beginning of a slot. When a packet arrives during a slot, the station senses the channel at the beginning of the next slot and operates according to the protocol.

The main characteristics of the CSMA/CD protocol [6] is now summarized in a point form.

- If the channel is sensed idle, for nonpersistent and 1-persistent CSMA/CD, the packet is transmitted, and for p-persistent CSMA/CD, the packet is transmitted with probability p, and is delayed by the end-to-end propagation delay, τ, with probability $(1 - p)$.

- If the channel is sensed busy, then, for nonpersistent CSMA/CD, the transmission is deferred for a certain amount of time and the algorithm is repeated. For 1-persistent CSMA/CD, the station defers until the channel is sensed idle and then transmits a packet. For p-persistent CSMA/CD, the station defers until the channel is idle and then, with probability p, transmits a packet, delaying transmission with probability $(1-p)$ by τ. In the latter case, after time

τ, the procedure is repeated.

- If a collision is sensed, the station aborts the packet being transmitted and transmits a jamming signal. The stations of the colliding packets then abort their transmissions and back off.
- At the time a backed-off packet is to be retransmitted, the algorithm is repeated.

Ethernet was developed by Xerox Corporation. The variant of CSMA/CD used is 1-persistent, nonslotted. According to [14] the protocol is as follows :

- A station acquiring a packet to transmit senses the bus. If the bus has been idle for an amount of time at least equal to the interpacket space, the station starts transmitting the packet.
- If the channel is sensed busy, the station defers until the channel is idle and then transmits the packet when the interpacket space is obeyed.
- If a collision is sensed, the station aborts the packet being transmitted and transmits a jamming signal. The stations of all colliding packets then abort their transmissions and wait for a random duration of time.
- At the time a packet is to be retransmitted, the algorithm is repeated.

The random duration of time is computed by the so-called backoff function. There are various functions that can be used: linear, exponential etc. In this research the exponential backoff function, $2\tau * U(0, 2^{cc})$, is used, where cc is the number of collisions suffered by the packet and U is a uniformly distributed random number from the specified interval.

2.1.2 Expressnet

Expressnet is a protocol that is designed to work on unidirectional buses such as optical buses. The system may be considered as consisting of two channels: the outbound channel which all stations access in order to transmit, and the inbound channel which stations access in order to read the transmitted information, Figure 2.1.

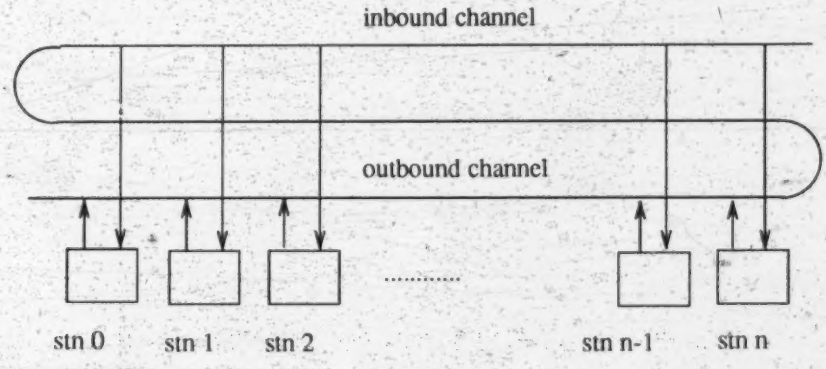


Figure 2.1 S-shape Expressnet.

In addition, stations can sense the activity at the outbound channel. A station senses the outbound channel and waits for EOC(OUT) (the end of carrier at the outbound channel). Immediately following the detection of EOC(OUT) it starts transmission of a preamble if it has a packet waiting for transmission. Simultaneously, it senses the outbound channel. The transmitting station immediately aborts the transmission if carrier from the upstream is detected. If the preamble has been transmitted without interference caused by a station at upstream, the station may assume that it has acquired the bus access and its packet will be sent successfully. The length of the preamble should be long enough to cover the longest detection

delay of the EOC(OUT) signal in the worst case. Otherwise, a station will be misled that it has gained the right to transmit and collision will occur. A succession of packets transmitted in a round is called a *train*. When the end of a train is detected and after a certain period of silence, a new round is started. This event is called EOT(IN). The length of the silent period is called the preamble delay, and it should be at least equal to or greater than the worst carrier detection delay. Each transmitted packet is separated by the length of the preamble and the carrier detection delay.

One problem is: how is a new round started? With one approach [20], [4], as soon as EOT(IN) is detected, a station which has a packet waiting for transmission transmits a burst for a period of time equal to two end-to-end propagation delays. The station will assume that it has obtained the right to transmit if during this period it has not heard any interference caused by upstream stations. Otherwise it will abort the transmission. However, in order to reduce the amount of time wasted, it is better to trigger the next round immediately by the EOT(IN) event and at the same time give higher priority to stations at the upstream. This can be achieved if the channel is arranged in certain ways, e.g. S-shape topology, Figure 2.1. Upon detection of EOT(IN) all alive stations, i.e. functioning stations, send a preamble even if they have no packet to transmit. This is to keep the network running and trigger another round to avoid a cold start-up of the network.

The preambles transmitted from other stations in response to EOT(IN) are overlapped by the preamble from the leftmost station. This is because the leftmost station hears the EOT(IN) first, and its preamble will arrive at other stations at about the same time as the EOT(IN). The trains are separated by a period of time

equal to $\tau + d_p + t_c$, where τ, d_p and t_c are respectively the end-to-end propagation delay, preamble delay, and the delay depending on the topology. In general the operation of this protocol can be described by the following rules:

- A station senses the outbound channel. When EOC(OUT) is detected and if it has a packet to transmit, it first transmits a preamble. It assumes that it has obtained the right to transmit if there is no interference. Otherwise it aborts the transmission and waits for EOC(OUT) again.
- Upon detection of EOT(IN), a new round is started. All active stations have to transmit the preamble to keep the network alive.

2.1.3 Token-Bus

In Token-Bus, the stations on the bus form a logical ring. That is, the stations assume an ordered sequence, with the last station of the sequence followed by the first. Each station knows the address of the station preceding and following it. The physical ordering of the stations on the bus is irrelevant and independent of the logical ordering. A control packet, known as the token, is passed around the logical ring to regulate the right of access to the bus. Token-Bus is far more complicated than CSMA/CD. It requires considerable network maintenance and management, yet on the other hand it provides more flexibility on the allocation of the right to access the bus. The maintenance and management include :

- **Initialization** When the network breaks down or starts up, some procedure is needed to start the network.

- **Addition** The network must provide a way for a nonparticipating station to insert itself into the logical ring.
- **Deletion** Same as the addition, a participating station should be able to remove itself from the ring.
- **Recovery** The network must be able to recover in case of logical error, e.g. duplicate token or lost token.
- **Priority** The Token-Bus scheme must provide flexibility on the allocation of the use of the bus.

IEEE 802.4 [14] defines a collection of procedures for the maintenance and management of the Token-Bus. A simplified model of Token-Bus is defined for this research. In this model, it is assumed that bus failure, token loss and duplication are unlikely to happen. All stations participate in the logical ring. No station ever wants to remove itself from the ring. Stations are assigned addresses 0, 1, 2, 3 ...etc. as shown in Figure 2.2. The token will be passed in the order of addresses. Station 0 starts the network by passing the token to its successor, station 1. The protocol of the Token-Bus in this thesis operates as follows :

- When a station receives the token, if it has a packet waiting for transmission, it transmits the packet. When the transmission is completed, the station releases the token and passes the token to its successor. When the station has no packet waiting for transmission, it immediately passes the token to its successor.
- There should be only one transmission per token.

- Packets and the token are transmitted after an interpacket space has been obeyed.
- When the network operates initially, station 0 is responsible for creating the token.

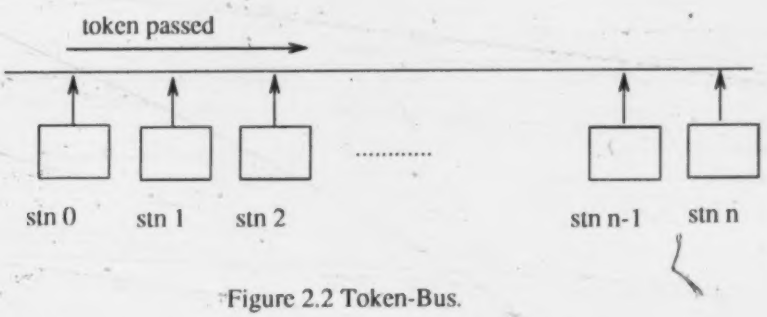


Figure 2.2 Token-Bus.

One slight change can be added to the above protocol of Token-Bus. In the above protocol of Token-Bus, a station can only transmit a packet when it holds the token. This can be changed in such a way that a station can hold the token and transmits more than one packet, when the number of messages waiting is more than a certain value. This Token-Bus is given the name, **Token-Priority**, to distinguish it from regular Token-Bus. The protocol of Token-Priority is summarized as follows:

- When a station receives the token, if there are messages waiting for transmission, it transmits the packet of a message. It then checks to see how many messages are lined up in the queue. If the number of the messages exceeds or is equal to a predefined value, the station has the priority to hold the token and transmits all the packets until the queue is clear. On the other hand, if the number of messages at the queue is less than the predefined value, the station passes the token. When the station has no packet waiting for trans-

mission, it immediately passes the token to its successor. The predefined value is arbitrarily set to two.

- Packets and the token are transmitted after an interpacket space has been obeyed.

2.1.4 CSMA/CD-DP

CSMA/CD-DP is a refinement of the CSMA/CD concept. It has two modes of operation, normal mode and synchronized mode. During normal mode, the operation of the network is exactly that of CSMA/CD. The difference is that after a collision all stations shift to synchronized mode. During synchronized mode, the right of access to the bus is synchronized by a "positive acknowledgment signal". This acknowledgment signal is usually a broadcasted message from the receiver, a packet end delimiter, or an end of carrier signal. The basic means to control the access right is a delay, the slot time, which is a multiple of the two-way end-to-end propagation delay of the network.

In synchronized mode, each of the stations in the network is assigned a transmission priority p_i which has a value between 0 and $(n-1)$, where n is the number of stations. A station has to wait for an amount of time (the so-called transmission delay) equal to $2\tau p_i$, where τ is the end-to-end propagation delay of the bus, for its turn to access the bus. If the station has a packet to transmit, it can then transmit the packet. If a transmission is detected before the station's slot arrives, it means that a station with a higher priority has transmitted a packet. In this case the current station will listen for the positive acknowledgment and then wait for the next slot. The priority number is cyclically changed after each positive acknowledgment

to guarantee that every station has a chance to transmit. For example, upon the reception of a positive acknowledgment, the value of p_i is updated to $(p_i - 1)$ *mod* n.

In our study it is assumed that failure of the bus and transmission errors will not occur, that is, packets always reach the destination in a good shape. Negative acknowledgment is not necessary. The protocol is outlined as follows:

- The network operates initially in the normal mode.
- Stations initially are assigned priorities according to the order in which they are on the bus. That is, station 0 has priority 0, station 1 has priority 1, and so on.
- During normal mode, when a station has a packet for transmission, it senses the bus, and, if it is idle, it transmits the packet after satisfying the interpacket space. If the bus is busy, the station waits until the bus is idle and transmits the packet after the interpacket space is obeyed. While the station is transmitting the packet, it keeps sensing the bus. If collision is heard, the station aborts the transmission and sends a jamming signal. It then waits for an amount of time equal to 2τ to make sure that the collision is over. After that, the station shifts to the synchronized mode.
- During synchronized mode, the synchronization of bus access is based on the detection of the end of carrier signal. A station waits for an amount of time equal to $2\tau p_i$ for its time slot. Meanwhile, it senses the bus. If it hears any activities, it means that another station with a higher priority has transmitted a packet. The station then waits for the end of carrier signal. Upon the

detection of the end of carrier, it updates its priority to $(p_i - 1) \ mod \ n$, and waits for its time slot again.

- All transmitted packets are preceded by the interpacket space.

2.1.5 General Network Parameters

In this thesis, all stations were equally distributed along a bus of length 240 bits-long bus. For the purpose of compatibility, the network parameters were based on the standard of Ethernet. The network parameters are listed as below:

- minimum packet length = 368 bits,
- maximum packet length = 12000 bits,
- header plus trailer = 208 bits,
- interpacket space = 96 bits (Ethernet, Token-Bus, & CSMA/CD-DP),
- jam length = 32 bits (Ethernet only),
- token length = 8 bits (Token-Bus only),
- preamble delay = 800 bits (Expressnet only) and
- preamble length = 8 bits (Expressnet only).

2.2 Performance Measures

For uncorrelated traffic, the average message delay is the performance measure of interest. In this thesis, the **message delay** [5] is defined as the amount of time

elapsed from the moment a message is queued for transmission to the moment that the entire message is successfully received at the destination. For RPC traffic and file server traffic, the average response delay is the measure of interest. The **response delay** is defined to be the amount of time between the moment that a request is put to the queue of the station for transmission and the moment in which the station receives the reply. **Throughput** is another quantity to measure performance. It is defined as the number of information bits received successfully at the destinations per time unit. The maximum value of throughput is called **channel capacity**. Usually throughput is normalized and the channel capacity is never greater than 1 for a bus network.

Different values of response delay, message delay and throughput are obtained by submitting different loads to the networks. For RPC and file server traffic, performance analysis is made by plotting response delay against throughput. For uncorrelated traffic, message delay is plotted against throughput to analyze performance. Suitable values of load are estimated, so as to have the full range of throughput values. Good performance of local area networks of RPC and file server are characterized by short average response delay and high channel capacity. For uncorrelated traffic, a good performance is indicated by short average message delay and a high channel capacity. The number of points used to draw a typical curve in the thesis is in the range of about 18 to 24. This number depends on the shape of the curve obtained. The number of messages generated during a single run to get a point is 30,000 messages. The time required to compute a point in any single run of the simulations is in the range of 450 to 20000 seconds. In this thesis, the throughputs of Ethernet for the three different kinds of traffic (Uncorrelated, RPC and File

Server traffic) saturate at about 0.6. Different levels of traffic loads are quantified approximately based on this value. The traffic load which gives throughput in the range below 0.2 is **light traffic**. The traffic load which gives throughput in the range of 0.2 to 0.4 is **medium traffic**. Any traffic load which gives the throughput value beyond 0.4 is considered to be **heavy traffic**.

2.3 Previous Results on the Performance of Selected LANS

2.3.1 Ethernet

The protocol of Ethernet is 1-persistent CSMA/CD. The results of some previous studies on the performance of CSMA/CD are briefly mentioned to review the performance of Ethernet. In [6], an analytical model of CSMA/CD is presented. In this model, it is assumed that:

- The number of stations is infinite.
- The arrival process from the infinite population is Poisson. The process includes both new and previously collided packets.
- All packets have the same length and the same transmission time.
- At any point in time, each station has at most one packet ready for transmission, including any possible previously collided packet.
- The channel is noiseless, so that transmission failures are due to collisions only.

Based on the above model, various relationships between throughputs and different *offered loads* are computed for nonpersistent, slotted nonpersistent and slotted 1-persistent CSMA/CD. The offered load g is defined as the average number of attempted transmissions during t seconds of channel time, where t is the packet transmission time. The throughput s is defined as the number of successful transmissions during t seconds. In each case, it is found that throughput decreases with increasing a, where a is equal to τ/t and τ is the end-to-end propagation delay. This is consistent with the intuition that as τ increases the time required for a station to detect a transmitting signal increases. There are very few analytical studies on the message delay for CSMA/CD. For slotted nonpersistent CSMA/CD, the analysis in [21] uses embedded Markov chains to obtain numerical solutions for the relationship of message delay and throughput.

In [16], the performance of CSMA/CD is also analyzed. In this study, time in the bus link is measured in slots. Each slot is equal to twice the end-to-end propagation delay. The number of active stations in the network is m. Each active station restrains itself to transmitting during a slot with probability p. There are two types of intervals in the bus, the **transmission interval**, and the **contention interval**. A contention interval is a sequence of slots, containing either a collision or a silence. A transmission interval is the number of slots required for a successful transmission. Time is normalized such that the packet transmission time equals 1 and the end-to-end propagation delay equals a, where a is τ/t. Thus, the transmission interval is equal to $1/2a$ slots. The throughput in this analysis is defined as:

$$s = \frac{transmission\ \ interval}{transmission\ \ interval\ +\ \ average\ \ contention\ \ interval}.$$

The transmission interval equals:

$$\frac{1}{2a}.$$

The average contention interval equals:

$$\sum_{i=1}^{\infty} i(1-A)A^i = \frac{1-A}{A},$$

where A is the probability that exactly one station attempts to transmit and acquire the bus link, and equals:

$$\binom{m}{1} p(1-p)^{m-1} = m * p(1-p)^{m-1}.$$

A has a maximum when p is equal to $1/m$:

$$A_{max} = (1 - \frac{1}{m})^{m-1}.$$

Thus, if this maximum value of the probability is assumed, the throughput equals:

$$s = \frac{1}{1 + \frac{2a(1-A_{max})}{A_{max}}}.$$

2.3.2 Token-Bus

In [16], the performance of Token-Bus is also studied. In this study, it is assumed that the token is passed in the order of the stations in the bus link, i.e. the logical ring is the same as the physical ring. The separation between two stations are same. Again, time is normalized. The packet transmission time equals 1, and the end-to-end propagation delay equals a. Throughput is defined as:

$$s = \frac{time\ spent\ in\ transmitting\ data\ in\ one\ cycle}{total\ time\ of\ one\ cycle}.$$

Also, a station, which has transmitted a packet, will wait for the return of the packet before releasing the token. So, there are two cases. In case one, a is less than or equal to 1 and in case two, a is greater than 1. In both cases, the time spent in transmitting data in one cycle is m, where m is the number of active stations. For case one, $a \leq 1$, the total time of one cycle is $m + a$. For case two, $a > 1$, the total time of one cycle is $ma + a$. The throughput is:

$$s = \begin{cases} \frac{1}{1+\frac{a}{m}} , & a \leq 1, \\ \frac{1}{a(1+\frac{1}{m})} , & a > 1. \end{cases}$$

This study [16] also reports the simulation results on the performance of Token-Bus and CSMA/CD [17]. In this simulation, two cases of message arrival are employed. In the first case, only 1 station out of 100 has messages to transmit and is always ready to transmit. In the second case, 100 stations out of 100 always have message to transmit. The result shows that the throughput of CSMA/CD decreases more when a is increased. CSMA/CD offers short message delay under light load, but long message delay under heavy load. Token-Bus offers long message delay under light load and relatively short message delay under heavy load.

2.3.3 Expressnet

Performance analysis of Expressnet appears in [20]. In this study, the throughput is evaluated as the ratio of the time required to transmit a train of data packets to the time separating the start of two consecutive trains. Time is again normalized. The packet transmission time is 1, and end-to-end propagation delay equals a, where a is τ/t. Given that m stations always have a message to transmit, the time required to transmit a train of data packets is m. The time separating the start of two

consecutive trains is $m + 2a$. The throughput is independent of the total number of stations and is equal to:

$$s = \frac{1}{1 + \frac{2a}{m}},$$

where $a = \frac{\tau}{t}$. In a model of infinite population, s is always equal to 1. This value is better than that of CSMA/CD. In order to attain a tractable analysis of packet delay and to compare Expressnet to CSMA/CD, [20] considers a model consisting of n stations each with a single packet buffer. A station is either idle or backlogged. An idle station generates a packet at a random time which is exponentially distributed. A backlogged station does not generate any packet and becomes idle upon successful transmission of the packet in its buffer. The packet transmission time is fixed. This model has been used previously to analyze CSMA/CD. The results of [20] indicate a superority of Expressnet over CSMA/CD.

2.3.4 CSMA/CD-DP

In [8] the performance of CSMA/CD-DP is analyzed using simulation and analytical methods. An exact analytical performance analysis does not seem to be feasible due to the complexity of the protocol. The following observations have been made, however, by assuming a Poisson arrival process. For light traffic (small a and n, where n is the number of stations), the approximation generally fits very well. For large overhead, the analytical results show increasing deviation from simulation. The result indicates that this protocol has high message delay in the medium load range. This delay is especially high when there is a large number of stations. This is due to the long transmission delay which equals $2\tau p_i$, where p_i is the transmission priority. But the protocol shows good performance at very low and heavy traffic.

No throughput analysis is found in [8]. However, it is not difficult to see that if heavy traffic is considered such that each station has a message to transmit at any point of time, this protocol can give a throughput of almost 1.

2.3.5 Summary

From the previous studies, the throughput value of the selected local area networks can be compared. The throughputs of the selected local area networks are listed below:

Token-Bus [16] : $s = \begin{cases} \frac{1}{1+\frac{a}{m}} & , \quad a \leq 1, \\ \frac{1}{a(1+\frac{1}{m})} & , \quad a > 1, \end{cases}$

CSMA/CD [16] : $s = \frac{1}{1+\frac{2a(1-A_{max})}{A_{max}}}$ and

Expressnet [20] : $s = \frac{1}{1+\frac{2a}{m}}$.

If the special case of m equal to infinity is considered, the throughput becomes:

Token-Bus : $s = \begin{cases} 1, & a \leq 1, \\ \frac{1}{a}, & a > 1, \end{cases}$

CSMA/CD : $s = \frac{1}{1+3.44a}$ and

Expressnet : $s = 1$.

The results indicate that Token-Bus and Expressnet have better maximum throughput than Ethernet. As mentioned in Section 2.3.4, in the extreme case of very heavy traffic, when every station has a packet waiting for transmission, the maximum throughput of CSMA/CD-DP almost reaches 1. However, the protocol has a very high message delay in the range of medium load traffic. In addition, it has short message delay at very low traffic and long message delay at heavy traffic.

The message delay of Token-Bus and Expressnet are short at heavy traffic, but long at low traffic as compared to CSMA/CD.

As will be observed in Chapter 3, the performance of the above protocols under uncorrelated traffic is similar to their performance under the traffic generated by a Poisson process given that the parameter of the uncorrelated traffic and the Poisson process are close to each other.

Chapter 3

Traffic Patterns

3.1 Uncorrelated Traffic

Uncorrelated traffic is characterized by the fact that a station independently has a message to transmit. At some moment, a station has a message to transmit, and after a while the station may have another message to transmit. This time interval is called the **message interarrival time**. The station does not need to suspend its work to wait for the reply from another station. This kind of traffic can be generated by assuming that the message interarrival time and the message length are governed by some probability density function. The exponential probability density function and the uniform probability density function are commonly used for this purpose:

$$\text{Exponential probability density function : } f(t) = \begin{cases} \frac{1}{mean} e^{\frac{t}{mean}} & , 0 \leq t < \infty, \\ 0 & , elsewhere. \end{cases}$$

$$\text{Uniform probability density function : } f(t) = \begin{cases} \frac{1}{max - min} & , min \leq t \leq max, \\ 0 & , elsewhere. \end{cases}$$

From the above density functions, the value of t (for examples, message length or message interarrival time) can be generated. For the uniform case, t is equal to $rnd * (max - min) + min$. For the exponential case, t is equal to $-mean * ln(1 - rnd)$, where rnd is a continuous random number in the range of [0, 1). Given the average values of the message interarrival time and the message length, traffic can be generated if the exponential distribution is used. Traffic generated in such a way depends totally on the statistical process, and does not show any characteristic of the network. Such traffic is uncorrelated.

3.2 Remote Procedure Call Traffic

Remote procedure call [2] provides a method of interprocess communication across a network. In a system with remote procedure call, a caller passes the parameters, the **request message**, across the network to the remote procedure, a callee, in a way similar to procedure calls in a program on a single machine. The caller suspends all its work until the remote procedure finishes its work and returns the result, the **reply message**, to the caller. Hence, the traffic generated by RPC is correlated in the sense that a station does not generate a message independently.

In a RPC system with file servers, large portions of traffic are concentrated at the file servers. If a RPC system has processes evenly distributed among stations, the traffic will be spread evenly throughout the network. The traffic pattern depends on the request message length, the reply message length, the remote procedure service time, and how often a remote procedure is called. There are not many publications which have the statistics for the above parameters. In [13] some measurements of the

investigation for the performance of *Firefly* RPC are reported. In this investigation, the caller packet and the result packet have the size of 74 bytes and 1514 bytes respectively. In [1], it is mentioned that 18366 remote RPC calls were made in a five-hour period, which is about one RPC call per second. Also in [1] it is stated that, in data transfers between procedures, the most frequently occurring calls transfer fewer than 50 bytes, and a majority transfer fewer than 200 bytes. The observation shows that the majority of procedures transfer only a small amount of data.

3.3 File Server

In some network systems, it is economical to designate one or more stations as **servers**, to provide file storage or printing services to other stations called **clients**. Information on file servers can be found in [10], [18], [3], and [19]. File server traffic is another type of correlated traffic. The traffic characteristics of a file server system is very similar to RPC. A client transmits a request to a server. In most cases, the client will suspend its work to wait for the reply. After the client has received the reply, it will then issue another request. For most file server systems, the traffic pattern depends on the following :

The Software and Hardware Overhead: This is the amount of time that a server takes to retrieve the data for the request. The software overhead mainly refers to the time taken by the server program. For example, in the Cambridge File Server (CFS) [10], the amount of time required, as pointed out, is about 11.5 millisecond. For Xerox Distributed File System (XDFS) [18], it is about 37.5 millisecond. The hardware overhead refers to the time spent by the disk to fetch and

transfer data. For example, the disk head access time of both CFS and XDFS is about 35 millisecond [18]. The disk transfer rate in CFS and XDFS is about 1 Mbyte/sec. Thus, in the worst case the combined software and hardware overhead of CFS is 47.5 millisecond and for XDFS is 73.5 sec to transfer 1000 bytes. However, in most cases, a file server has a cache to speed up the data transfer process. This software and hardware overhead constitutes the amount of time needed to serve the request from a client. A short hardware and software overhead gives a short amount of time that a client has to wait for the reply.

Length of Request and Reply: Most requests consist of a code to specify the action required, and the parameters. For example:-

- read(fileUID,offset,length).
- write(fileUID,offset,length, a series of bytes).

The reply message from a server is a series of bytes read from the file, a short message telling what is happening, or just a file identifier. The unit of data transfer is either a block or a whole file. Each approach has its own advantages and disadvantages. Whole file transfer reduces the traffic load; this is reported in [7]. In [12] it is shown that about 70% of all file accesses are whole file transfers and that about 80% of all files accessed are less than 10 Kbytes long. It is reported in [11] that about 99% of files are less than 64K and the average file size is 1K bytes. All these statistics support the statement that most files are small in size and are accessed as a whole.

The Request Behavior: The request behavior depends on the following factors:

- Does a client suspend any further requests after one request has been issued and before the response arrives?
- Does a request involve multiple clients and multiple servers? Some file server systems support requests involving multiple servers and multiple clients [19], [18].
- What is the handshaking protocol used between the client and the server?

There is no restriction on whether or not another request can be issued before a reply is received. Another request prior to the reception of a reply to the previous request is possible in [18]. Each reply message specifies the identity of the request to which it responds, so that the client can match replies with requests. This permits clients to initiate new requests without waiting for previous ones to be completed. Whether or not this parallelism is necessary depends on the clients, since the server does not guarantee the order of the replies.

In general [3] [18], a client initiates a transaction by sending an '*open transaction*' request to the server. Upon receiving a transaction identifier sent from the server, a client can perform the appropriate operations – reading, writing , or deleting a file. When the transaction is finished, the client sends '*close transaction*' to inform the server. There are two main types of client and server protocols. In one type of protocol, a client acknowledges the server [19]. For the other kind of protocol, a client simply does not acknowledge the server. This kind of protocol [3] relies on the reliability of the network.

Local Cache: In some cases, a client may have a local cache, which reduces the demand on transmission. Upon any transaction, a client may first consult its local

cache for the file it wants. If the file is not available in the local cache, then it has to send the request to the servers.

3.4 Network Load

In general, the load on a network depends on the number of stations, the message length, and how often messages are transmitted. Moreover, the load of a network also depends on other parameters of the traffic pattern. The network load of uncorrelated traffic, RPC traffic and file server traffic is described as follows:

Load of Uncorrelated Traffic: The load of a network under uncorrelated traffic depends on the average message interarrival time, the average message length and the number of stations. As mentioned before, in uncorrelated traffic, a station independently generates a message to transmit. Unlike RPC and file server traffic, a station has a message to transmit at some moment, and after some time, the station may have another message to transmit.

Load of RPC Traffic: The load of a network under RPC traffic depends on the number of stations, the average request length and the average reply length. Here, a station does not independently have a message to transmit; a process transmits a request and then suspends all its work and waits for the reply. After the process receives the reply, another request may occur. **Request interarrival time** is the amount of time between the moment that the process receives the reply and the moment that the process has another request. This request interarrival time also contributes to the network load of RPC traffic. The number of processes in a station

is also a factor of the network load because there will be more demands for the bus if there are more processes in a station. The network load also depends on the amount of time that a process needed to serve a call.

Load of File Server Traffic: Similar to RPC traffic, the load of the network under file server traffic depends on the average request length and the average reply length. In addition, the load depends on the number of clients and servers. The load also depends on the **request interarrival time**. This is the amount of time between the moment a client receives a reply to the moment the client generates another request. The service time that a server needs to serve a client request also contributes the network load. Short request interarrival time and service time result in a heavy traffic.

Chapter 4

Performance Under Uncorrelated Traffic

4.1 An Uncorrelated Traffic Model

The performance of Ethernet, Expressnet, Token-Bus and CSMA/CD-DP is reviewed under the uncorrelated traffic pattern. Traffic is generated by assigning a traffic generator to each station. An exponential distribution is used. The message interarrival time and the message length are computed according to this distribution. Given an average value of the message interarrival time and the message length, messages can be generated. The traffic generator computes the message length and inserts the message to the message queue of the station. The traffic generator then calculates the message interarrival time, and sleeps for this amount of time. It generates another message when it wakes up. The station picks up the message and transmits the message according to the protocol. In this thesis, the

average message interarrival time is the parameter used to produce different traffic intensities. The average message length is arbitrarily set to 2000 bits.

4.2 Simulation

The number of stations is another network parameter. Sets of simulations are run on each network (Ethernet, Expressnet, Token-Bus, Token-Priority and CSMA/CD-DP). In each set, simulations are run on the networks with 11, 41, 81 and 121 stations. Different values of average message interarrival time are submitted to the network to get a full range of throughput. Performance is studied by plotting message delay against throughput.

4.3 Performance Comparison

Figure 4.1 - 4.4 show the results of the average message delay against throughput. From these plots, the performance of Ethernet, Expressnet, Token-Bus and CSMA/CD-DP is reviewed and the performance of Token-Priority is compared with Token-Bus.

4.3.1 Ethernet

Ethernet performs best at low traffic. It has the shortest average message delay. This is easy to explain, because the protocol of Ethernet has no overhead to transmit a packet, unless the packet collides with other packets. Other local area networks, for examples Token-Bus and CSMA/CD-DP (in synchronized mode), have to wait

for the token or their time slot before they can transmit their packet. As the traffic becomes more heavy, other local area networks show lower average message delays, because collisions are likely to happen and stations have to abort transmission and wait for a random amount of end-to-end propagation delay before transmitting again. The average message delay of Ethernet is unbound. This is contrary to Token-Bus, CSMA/CD-DP and Expressnet. However at some loads of moderate traffic when there are few stations (e.g. 11 stations), the performance of Ethernet shows some improvement, as indicated by a shorter average message delay (Figure 4.1 - 4.2). This can be explained by the following: When the traffic is moderate, two transmitted packets, for example, from stations A and B, collide with each other. They have to backoff and wait for some time before they can try again. Meanwhile, when these colliding stations are waiting for the backoff time to finish, a third station, say C, will have a very high probability of transmitting a packet successfully. This improvement disappears when there are more stations, because station C will have to compete with more stations to transmit a packet, and its transmitted packet has a higher probability of colliding with other packets.

4.3.2 Token-Bus

At low traffic Token-Bus has a longer average message delay than Ethernet. This is because a station has to receive the token before it can start the transmission. This discrepancy between the performance of Ethernet and Token-Bus is even more obvious when there are more stations, since a transmitting station has to wait longer for the token arrival. If it is assumed that the token is passed to an immediate neighbour, and there are m stations on the average which have a packet waiting for

transmission, then the amount of time, the **average round length**, that a station has to wait after its last right of access to the bus, can be derived as shown :

Let

- n = total number of stations in the network,
- m = number of stations on the average that have packets waiting for transmission in a round,
- p_l = average packet size,
- τ = end-to-end propagation delay,
- p_t = token size, and
- p_s = interpacket space.

Since there are, on the average, m packets waiting for transmission, the amount of time spent waiting due to m packets is $m(p_l + p_s)$. The token is passed around all stations. This amount of time is $n(p_t + p_s)$. And the token travels a distance of 2τ. Thus the average round length is :

$$m(p_l + p_s) + n(p_t + p_s) + 2\tau \ .$$

This average round length indicates that when there are few packets, for example 1 or 2, a station has to wait a long time for its turn to transmit a packet; while in the case of Ethernet, under such light traffic, it is very possible that the transmission is successful without any collisions. During very heavy traffic, there are more packets queued at stations, but the packet delay is bounded by $n(p_{lmax} + p_t + 2p_s) + 2\tau$,

where p_{lmax} is the maximum packet length. In the case of Ethernet, however, the average message delay is unbounded.

4.3.3 Token-Priority

Token-Priority shows good performance when the traffic is extremely heavy. In uncorrelated traffic, messages are generated at intervals. An interval is computed using exponential distribution by assuming an average value and is independent of other stations. If this value is small, the traffic will be very heavy, and more messages wait for transmission at each station. With regular Token-Bus, in each round a station is allowed to transmit only one packet. But in Token-Priority, a station can transmit more than one packet and clear its queue when it is holding the token. The result shows that Token-Priority has a comparatively shorter message delay than Token-Bus.

4.3.4 CSMA/CD-DP

CSMA/CD-DP has the longest average message delay at low and moderate traffic, especially when there are more stations. Such a long average message delay can be explained by the following: At very low traffic, with few ready stations, collision is not likely to happen; then CSMA/CD-DP performance is similar to Ethernet. But at moderate traffic, collisions occur, and CSMA/CD-DP changes to synchronized mode. As mentioned in Section 2.1.4, during synchronized mode, stations are waiting cyclically in each round for their slots to transmit.

The amount of time a station has to wait after a successful transmission is detected equals $2\tau p_i$, where p_i is the priority number and τ is the propagation

delay. For example, suppose that during a cycle there is only one station which has a packet waiting for transmission. In the worst case, the transmission delay (i.e. the amount of time a station has to wait for its slot) is $2(n-1)\tau$, where n is the number of stations. This worst case occurs when the priority of the station is the lowest. In the best case, the station has the highest priority, and it transmits the packet right away. Therefore, the average transmission delay that a packet has to wait is about $(n-1)\tau$. This causes a very high average message delay for medium traffic especially when there are more stations in the network. In a similar case, for Token-Bus, a station only needs to wait for a maximum of $2\tau + n(p_l + p_s)$, which is much less than that of CSMA/CD-DP. But when the traffic is very heavy, the performance of CSMA/CD-DP improves and is even better than the performance of Token-Bus. During very heavy traffic, it is likely that each station has a packet waiting for transmission. Stations transmit packets according to their priority number after an end-of-transmission is detected. The average round length is equal to $n(p_l + p_s)$ where n, p_l and p_s are the number of stations, the average packet size and the interpacket space respectively. For the same heavy traffic, the average round length of Token-Bus is $n(p_l + p_l + 2p_s) + 2\tau$ This average round length is longer than that of CSMA/CD-DP, and it can explain why CSMA/CD-DP performs better during very heavy traffic.

4.3.5 EXPRESSNET

Expressnet outperforms the other networks, even though Ethernet has better performance at very low traffic. Stations take turns to transmit packets with the upstream stations having the highest priority. Every station has a chance to transmit in each

round. If a transmitting packet is interrupted by a packet from upstream, it can transmit again after the transmission of the packet from upstream is completed.

The average round length of Expressnet is derived as follows :

Let

- m = average number of packets waiting for transmission in a round,
- p_l = average packet size,
- τ = end-to-end propagation delay,
- p_p = preamble size, and
- d_p = preamble delay.

Since each packet is accompanied by a preamble, it takes the amount of time, $m(p_l + p_p)$, just to insert m packets to the bus link. The packets and the preamble travel a distance of 2τ. The allowed period of silence, the preamble delay, after the end of a train is detected is d_p. Thus the average round length is:

$$m(p_l + p_p) + 2\tau + d_p .$$

When this average round length is compared to those of Token-Bus and CSMA/CD-DP, it can be found that Expressnet has a shorter round length. This means that a station in Expressnet waits a shorter time to transmit the next packet. Thus this explains why Expressnet has better performance than other local area networks.

4.4 Conclusion

Under the uncorrelated traffic pattern, Ethernet has the best performance at very low traffic, but poor performance at heavy traffic. It performs better than Token-Bus and CSMA/CD-DP when the traffic is moderate. This is especially obvious when there are more stations. CSMA/CD-DP has better performance than Ethernet at heavy traffic, and better than Token-Bus at very heavy traffic. The performance of Token-Bus is better than Ethernet at heavy traffic. At very heavy traffic, Token-Priority has better performance than Token-Bus. Expressnet has the best performance, except for very low traffic when it is marginally worse than Ethernet.

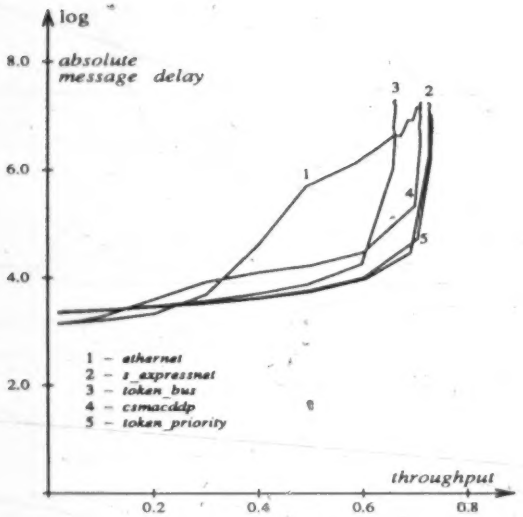


Figure 4.1, Uncorrelated traffic, 11 stations
average absolute message delay vs throughput

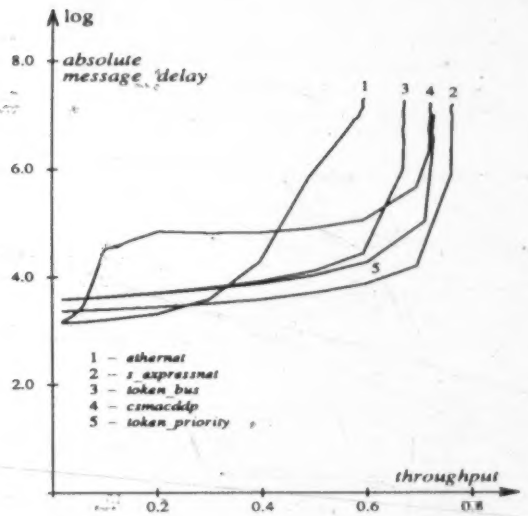


Figure 4.2, Uncorrelated traffic, 41 stations
average absolute message delay vs throughput

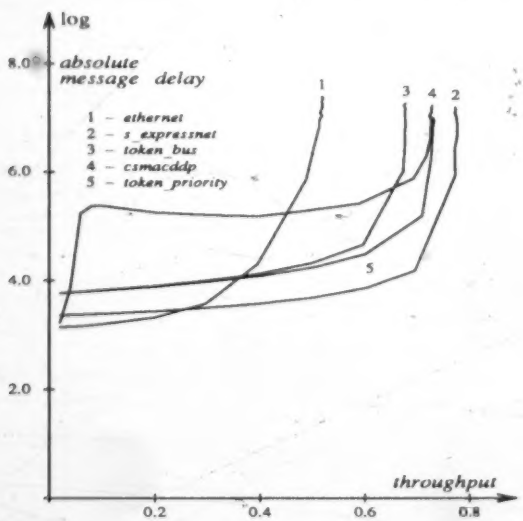


Figure 4.3, Uncorrelated traffic, 81 stations
average absolute message delay vs throughput

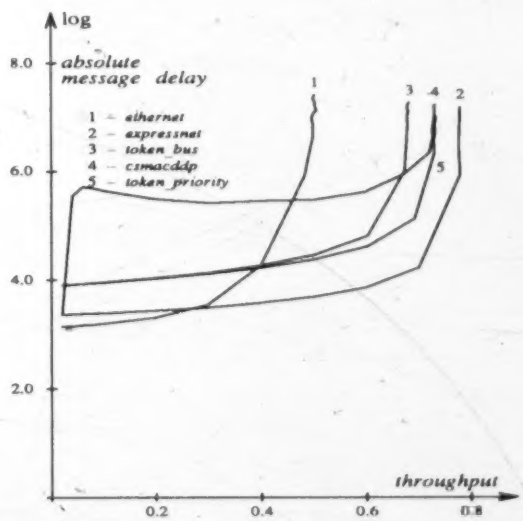


Figure 4.4, Uncorrelated traffic, 121 stations
average absolute message delay vs throughput

Chapter 5

Performance Under Remote Procedure Call Traffic

5.1 Model

In this model, each station can have one or more processes. Each process can be a caller which sends procedure parameters (the request message) to another process, a callee. The callee sends results (the reply message) back to the caller. A process is both a caller and a callee. That is, it sends and also receives request message. When a process sends out a request message, it waits until the reply message is received. In this model, a reply has higher priority than a request. This means that when a station has a reply and a request waiting for transmission in a queue, the reply will be transmitted before the request. The reply message length, request message length and the request interarrival time are assumed to be exponentially distributed. The average request interarrival time is the network parameter used to

produce different traffic intensities. When a callee process receives a request it goes to sleep for an amount of time to simulate the service time required to compute the result for the request of the caller. When the callee wakes up, it generates the reply and transmits it to the caller. When the caller receives the reply, it generates another request using the same exponential distribution.

From Section 3.2, it has already been indicated that most data transfers by an RPC call are short. In this model, the request message is assumed to have the average of 256 bytes. For the reply message, since it is assumed that a block of data may be transferred (for example, an array of characters or records), the reply message is given an average of 1024 bytes. The time that the remote procedure takes to process a call is arbitrarily assigned to have an average of 0.5 millisecond also distributed exponentially.

5.2 Simulation

Two cases of RPC are investigated. In the first case, each station has a single process only. When the process has sent out a request, it suspends transmission of any further request, until it has received the reply. In the second case, each station has three processes. Once a process has sent a request, it suspends transmitting any more requests until it has received the reply from its callee. However, other processes may send requests to their callees in the meantime. Sets of simulations are run on each local area network (Ethernet, Token-Bus, CSMA/CD-DP, and Expressnet) for each of the two above cases. In each set, simulations are run with the number of stations equal to 11, 41, 81, 121. Extra simulations is also run on Token-Priority

for each case of RPC (one process and three processes) with 81 stations to compare the performance of Token-Bus against the performance of Token-Priority under this traffic pattern.

5.3 Observations

The results of case one (stations with one process) are shown in Figure 5.1- 5.4. Figure 5.5 - 5.8 show the same results of Figure 5.1 - 5.4, but only lower values of the response delay (below $2 * 10^5$) are plotted. The results of case two (stations with three processes) are shown in Figure 5.9 - 5.12. Figure 5.13 - 5.16 also show the same results of Figure 5.9 - 5.12, again with only lower values of response delay (below $2 * 10^5$) plotted. The reason for plotting the results twice is to show clearly the performance of the local area network at low traffic intensities.

5.3.1 Single Process

CSMA/CD-DP, at low and moderate traffic, has the longest average response delay. At extremely low traffic, CSMA/CD-DP behaves like Ethernet, because collision is not likely to happen, and so CSMA/CD-DP operates in the normal mode. At low and moderate traffic, when a collision occurs, CSMA/CD-DP operates in the synchronized mode; the average response delay increases rapidly and goes up again at very heavy traffic. This is illustrated in Figure 5.1 - 5.4. In general, the results are very similar to those of uncorrelated traffic; Ethernet has the shortest response delay at very low traffic, (Figure 5.5 - 5.8). Token-bus, at busy traffic, performs better than CSMA/CD-DP and Ethernet by having shorter average response delay

and higher channel capacity. Expressnet still has the shortest average response delay at moderate and busy traffic, and the highest channel capacity.

5.3.2 Three Process

In the case that each station has three processes, generally, the results do not show too much difference from that of a single process. The traffic in this case is heavier. Higher response delays are observed. CSMA/CD-DP still has the longest average response delay at low and moderate traffic (Figure 5.9 - 5.12). Expressnet dominates other local networks. Ethernet has slightly better performance by having a shorter response delay at low traffic than others (Figure 5.13 - 5.16).

5.4 Explanation of the Observation

The protocol of Ethernet has the least overhead for transmitting a packet. This is the reason that it has the best performance at very low traffic. At busy traffic, collisions are very likely to occur, and the collision-free Token-Bus performs better than Ethernet. Unlike the case of uncorrelated traffic pattern, the intensity of the traffic depends more on the response delay. As indicated in Section 4.3.4, a station in CSMA/CD-DP has to wait a long time for its turn to transmit, about $(n-1)\tau$, during moderate traffic. The longer this delay, the higher will be the response delay. Consequently, less traffic is generated which results in a low throughput, as compared to Token-Bus and Expressnet when the traffic is a low and moderate. This explains why CSMA/CD-DP has a higher response delay and low channel capacity. However, similar to the results of uncorrelated traffic pattern, CSMA/CD-DP has

better performance when the traffic is very heavy. During very heavy traffic, when it is very likely that every station has at least one packet waiting for transmission, the average round length is $n(t_p + t_s)$. This amount of time is shorter than that of Token-Bus. In a remote procedure call system, very heavy traffic will be produced when there are more processes in a station. This better performance of CSMA/CD-DP is indicated in Figure 5.9 - 5.12. Expressnet has the best performance; this is due to the reason that it has the shortest average transmission round length. Stations in Expressnet wait a shorter length of time to transmit the next packet. Token-Priority and Token-Bus both show very similar performance under the RPC traffic (Figure 5.3, Figure 5.7, Figure 5.11, Figure 5.15). In a system of RPC, traffic is unlikely to be generated according to a simple statistical distribution. A message is only generated after the calling process has received the response. So, it is not likely that stations have a long queue of messages. Thus it is expected that the performance of Token-Bus and Token-Priority is similar.

5.5 Conclusion

Results show that for RPC traffic, Ethernet, Token-Bus, CSMA/CD-DP and Expressnet have performance similar to uncorrelated traffic pattern. CSMA/CD-DP has the highest response delay at moderate traffic. During very heavy traffic, when there are more processes in a station, CSMA/CD-DP shows a better performance similar to the situation of the uncorrelated traffic pattern in which large number of messages are generated from the statistical process.

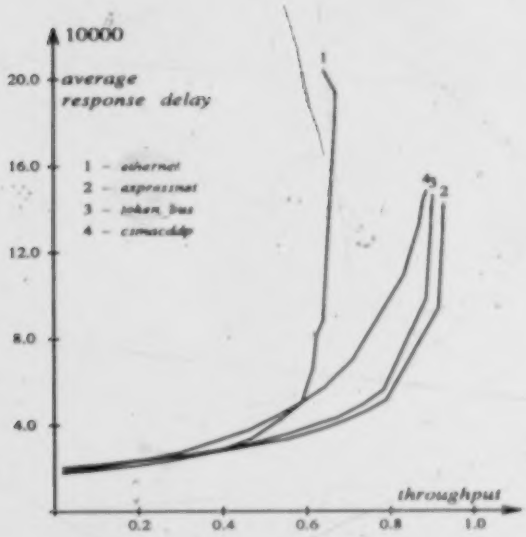


Figure 5.1 RPC average response delay vs throughput, 11 stations, 1 process

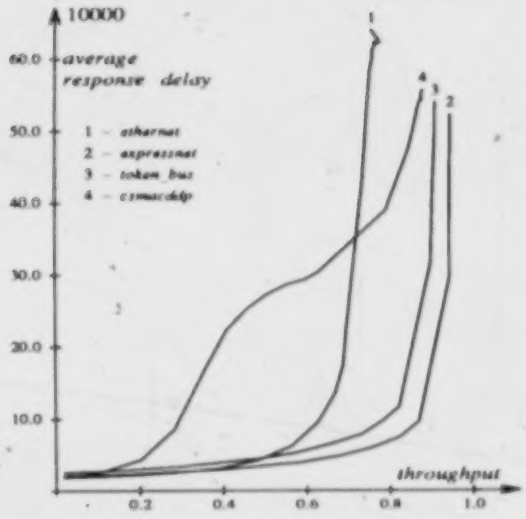


Figure 5.2 RPC average response delay vs throughput, 41 stations, 1 process

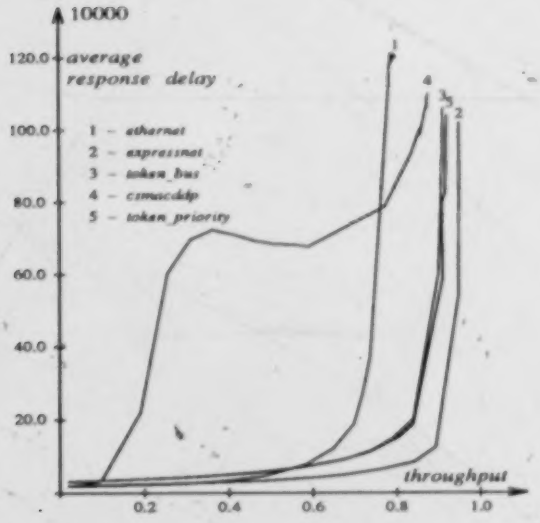


Figure 5.3 RPC average response delay vs throughput, 81 stations, 1 process

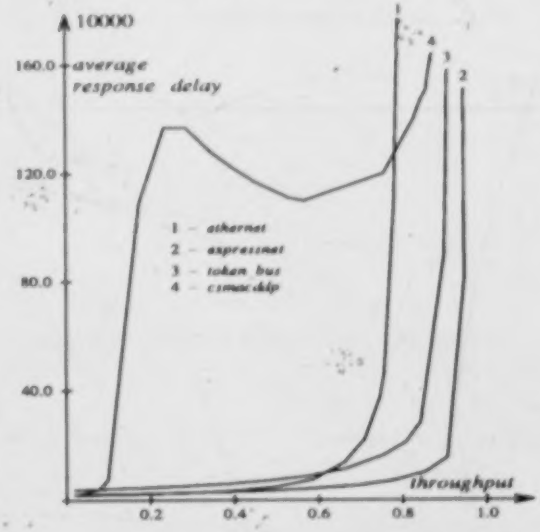


Figure 5.4 RPC average response delay vs throughput, 121 stations, 1 process

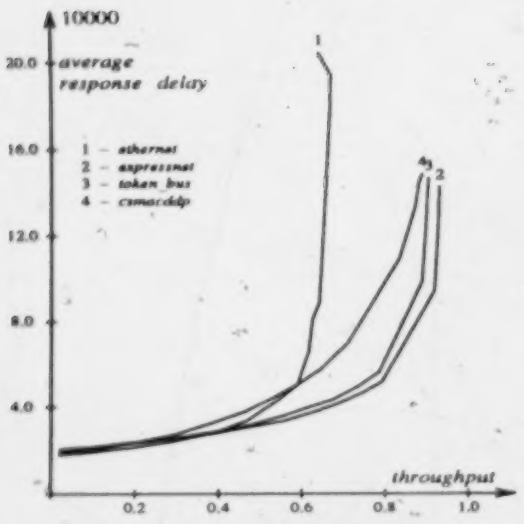


Figure 5.5 RPC average response delay vs throughput, 11 stations, 1 process

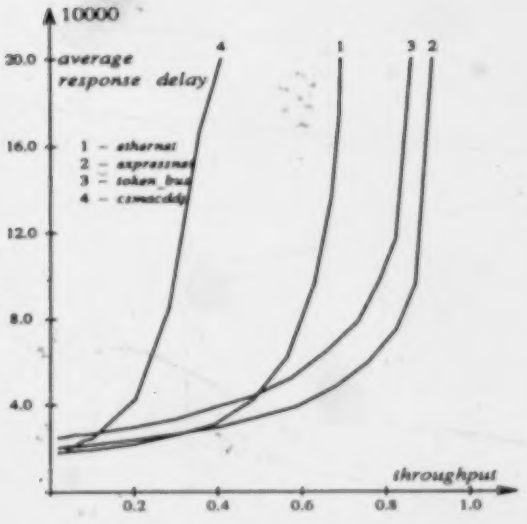


Figure 5.6 RPC average response delay vs throughput, 41 stations, 1 process

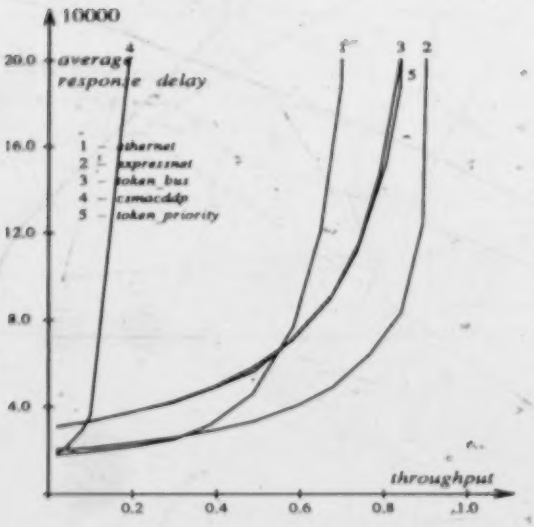


Figure 5.7 RPC average response delay vs throughput, 81 stations, 1 process

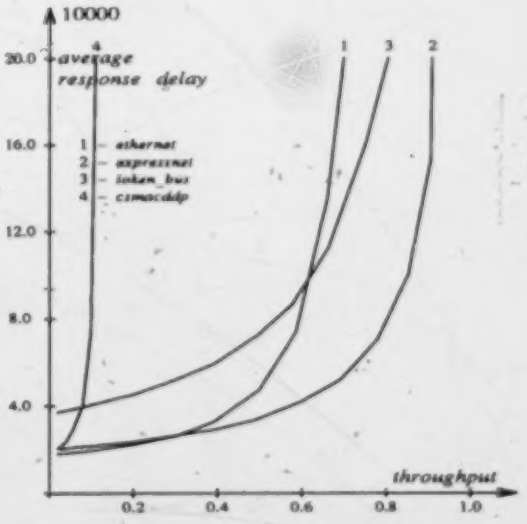


Figure 5.8 RPC average response delay vs throughput, 121 stations, 1 process

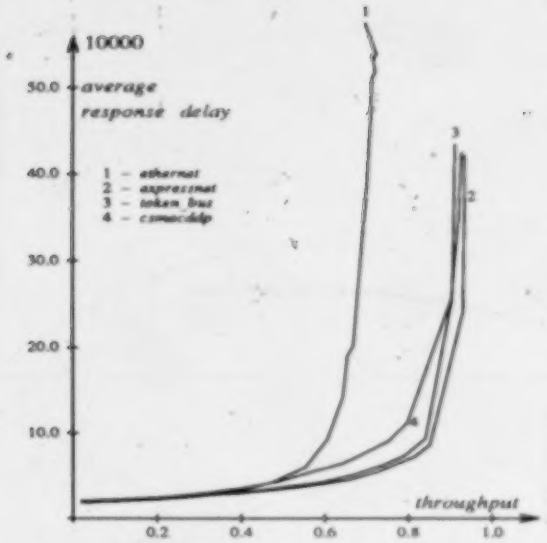


Figure 5.9 average response delay vs throughput, 11 stations, 3 processes

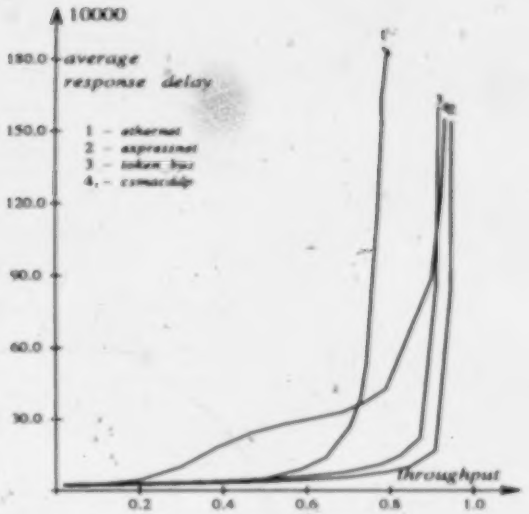


Figure 5.10 average response delay vs throughput, 41 stations, 3 processes

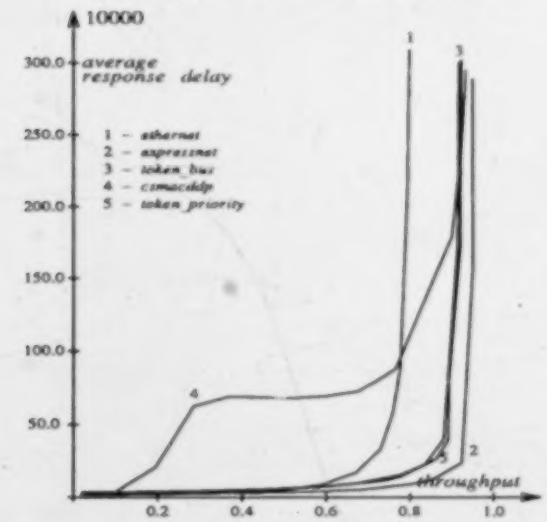


Figure 5.11 average response delay vs throughput, 81 stations, 3 processes

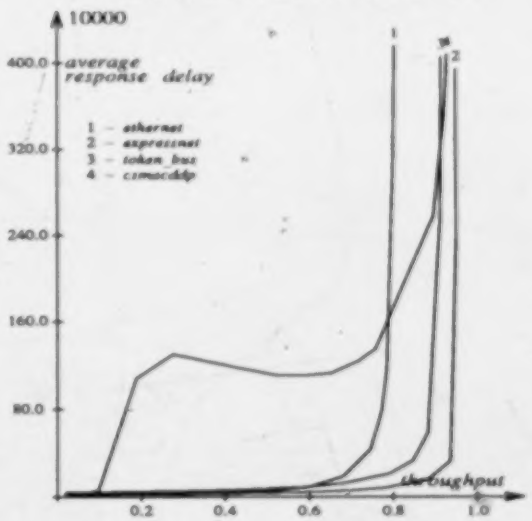


Figure 5.12 average response delay vs throughput, 121 stations, 3 processes

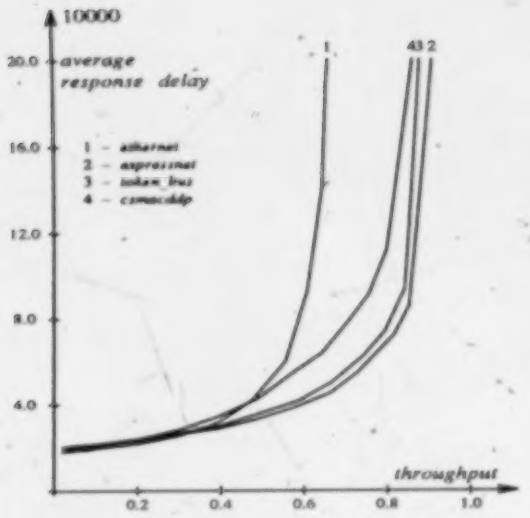


Figure 5.13 average response delay vs throughput, 11 stations, 3 processes

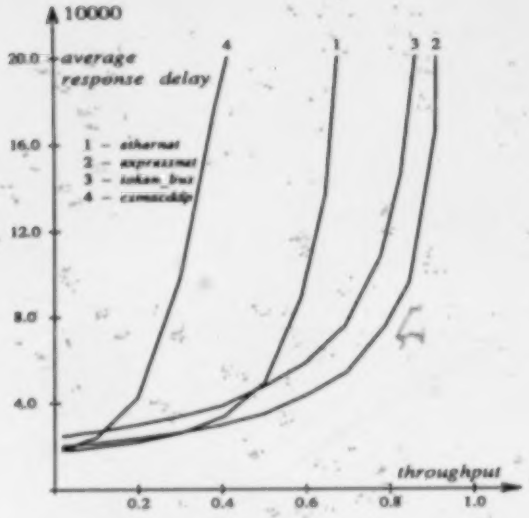


Figure 5.14 average response delay vs throughput, 41 stations, 3 processes

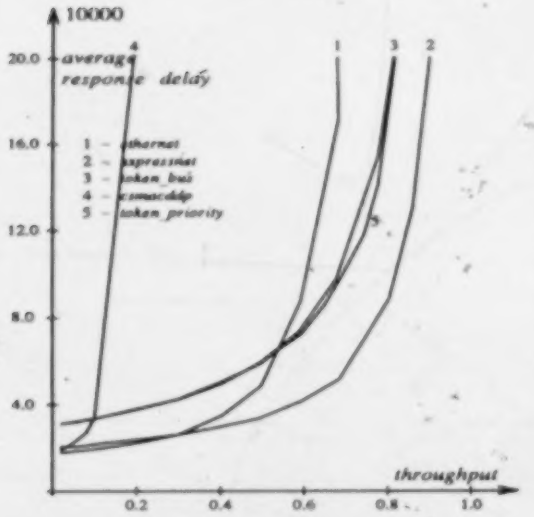


Figure 5.15 average response delay vs throughput, 81 stations, 3 processes

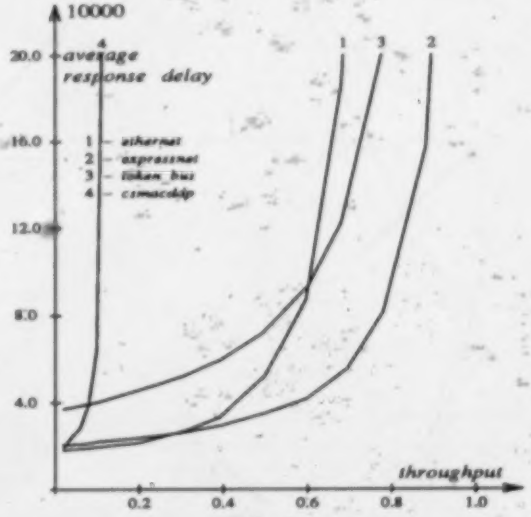


Figure 5.16 average response delay vs throughput, 121 stations, 3 processes

Chapter 6

Performance Under File Server Traffic

6.1 A File Server Model

A file server model is defined by considering the software and the hardware overhead of a file server system, the length of request and reply messages, the request behavior and whether or not a cache is present.

6.1.1 Software and Hardware Overhead

It is assumed that the file server is running with a cache to provide fast file transfer. The hardware overhead can be about 0.5 millisecond. The software overhead is assumed to be about 0.5 millisecond. This latter number is chosen to be the same as that of RPC. The average total overhead, the combined software and hardware overhead is exponentially distributed with an average of 1 millisecond.

6.1.2 Request and Reply

The requests of a client and the corresponding replies of the model are shown as below :

ReadFile(FID) —— ReadAck(FID,bytes)

WriteFile(FID,bytes) —— WriteAck(FID)

'WriteFile(FID,bytes)' and 'ReadAck(FID,bytes)', in this file server model, have an average length of 1024 bytes, distributed exponentially. Both, 'ReadFile(FID)' and 'WriteAck(FID)' have a fixed length of 256 bytes. The ratio of writes to reads is 1 to 4.

6.1.3 Request Behaviour

A client will keep on waiting until it receives a reply; no timeout is set. A client does not send acknowledgment to the server upon receiving a reply. There is no parallelism of multiple requests from a client. When a client has issued a request, it will wait for a reply. Meanwhile, it suspends all its work; no other request will be issued until it has received a reply. Also, no transactions involve multiple clients and multiple servers. Each request has exactly one reply. In this file server model, a request generator is assigned to each client and a reply generator is assigned to each file server. Based on the write to read ratio (1:4), a request is generated at a client station. The length of the request is generated according to an exponential distribution. If the request is a 'ReadFile' request, the average request length is 256 bytes. If the request is a 'WriteFile' request, the average request length is 1024 bytes. When the server receives a request, it checks if the request is a 'WriteFile'

request or a 'ReadFile' request. Then the server computes the service time (average is 1.0 millisecond) using the exponential distribution and simulates the service by sleeping for this amount of time. After that, a reply is generated. The length of the reply is calculated from the exponential distribution. If the reply is a 'ReadAck', the average reply length is 1024 bytes. If the reply is a 'WriteAck', the average reply length is 256 bytes. While the client is waiting for a reply, it suspends all its work. When a client has received a reply, the client can generate another request. The average request interarrival time is the network parameter to produce different traffic intensities, and the request interarrival time is computed from the exponential distribution. Response delay versus throughput is plotted to investigate the network performance under the file server traffic pattern.

6.2 Simulation

There are several points which are worth investigating. Point 1: how do various local area networks perform with the same number of servers at fixed location? Point 2: how does a local area network behave with different number of servers? Point 3: how do locations of servers affect the performance of a network? and Point 4: how does Token-Priority perform as compared to Token-Bus under file server traffic? To answer the above questions, the following simulation experiments are implemented as described below.

Point 1: Performance Comparison Among Various Networks Sets of simulations are run on each local area network. In each set, simulations are run on cases with 1, 2 and 3 servers located at the center of the bus. The number of stations

for each set of simulations is 11, 41, 81 and 121. Extra simulations are run on each local area network with cases of 5, 7, 10, 20, 30, 40 servers at the center of the bus. The number of stations is fixed at 81.

Point 2: Performance of a Network with Different Numbers of Servers For point 2, some of the results from point 1 are used. They are the results of the simulations on each network with cases of 1, 2, 3, 5, 7, 10, 20, 30, and 40 servers at the center of the bus and the number of stations fixed at 81.

Point 3: Performance with Servers at Various Locations Two locations are considered, the center and the end of the bus. Some of the results from point 1 are used (simulations on each network with 1, 3, 10, and 30 servers at the center and the number of stations fixed at 81). Extra simulations are run on each network with 1, 3, 10, and 30 servers at the right end of the bus with the number of stations fixed at 81.

Point 4: Performance Comparison of Token-Priority and Token-Bus Simulations are run on Token-Priority with cases of 1, 3, 10, 20, 30, 40 servers at the center and the number of stations fixed at 81. Results of the same simulations for Token-Bus are obtained from point 1 to do the comparison.

6.3 Observations and Explanation

6.3.1 Performance Comparison Among Various Networks

Performances of Ethernet, Expressnet, Token-Bus, CSMA/CD-DP are compared to each other in the cases of 1, 2, 3, 10, 20, 30, and 40 servers located at the center of the bus. In the case of 1 server, and 41 stations (Figure 6.1 - 6.4), Ethernet has shorter average response delay and higher channel capacity (maximum throughput) than Token-Bus and CSMA/CD-DP as indicated by the figures. For the case of 2 servers (Figure 6.7 - 6.8), Ethernet shows better performance by having lower response delay and higher channel capacity than CSMA/CD-DP. CSMA/CD-DP has the worst performance. Also Ethernet performs better than Token-Bus when the number of stations is more than 41. But when there are 3 servers (Figure 6.9 - 6.12), Ethernet still has better performance than CSMA/CD-DP. However, Ethernet performs better than Token-Bus only when the number of stations is more than about 81 stations. In the cases of 5, 7, 10, 20, 30, and 40 servers with 81 stations (Figure 6.13 - 6.18), the performance of Ethernet remains better than that of CSMA/CD-DP. But Token-Bus has better performance than Ethernet.

The observations show that when there are few stations, Ethernet has better performance than CSMA/CD-DP and Token-Bus. There are two possible explanations for this. First, Ethernet has the least overhead to transmit a packet. Second, the file server traffic is concentrated. This means that the server has the biggest load. A packet has a better chance of reaching the destination successfully because the population of contenders is small which results in a smaller probability of collision. On the other hand, concentrated traffic has no advantages for CSMA/CD-DP

and Token-Bus. In the case of Token-Bus each reply queued at the server has to wait for exactly the average round length before it is transmitted. For CSMA/CD-DP, the situation is even worse. The high response delay of CSMA/CD-DP can be explained as follows:

Assume that there is only 1 station which has a queue of packets waiting for transmission. During synchronized mode, in one cycle the station can transmit n packets. The total transmission delay is $1*2\tau+2*2\tau+3*2\tau+4*2\tau+.....+(n-1)*2\tau$, where n and τ are the number of stations and the end-to-end propagation delay. That is about $n\tau/2$ per packet on the average. This scenario can be applied to a server system where most of the traffic is concentrated at the server. This can explain why CSMA/CD-DP has a very high response delay.

As there are more servers, the load of each server is comparatively less and a reply has to wait a shorter time for transmission. Thus the performance of Token-Bus can improve when there are more servers. For the case of Ethernet, as there are more servers, the traffic is more widely spread among the servers. Collisions are very likely to occur and results in deterioration of performance.

6.3.2 Performance with Different Numbers of Servers

Performance of each local area network, Ethernet, Token-Bus, Expressnet, and CSMA/CD-DP is analyzed with respect to a varying number of servers. Response delay is plotted against the average request interarrival time, because when throughput reaches the channel capacity, any difference in performance is not easy to see if throughput is plotted (Figure 6.19 - 6.22). The results show that Token-Bus, Expressnet, and CSMA/CD-DP have better performance with more servers. When

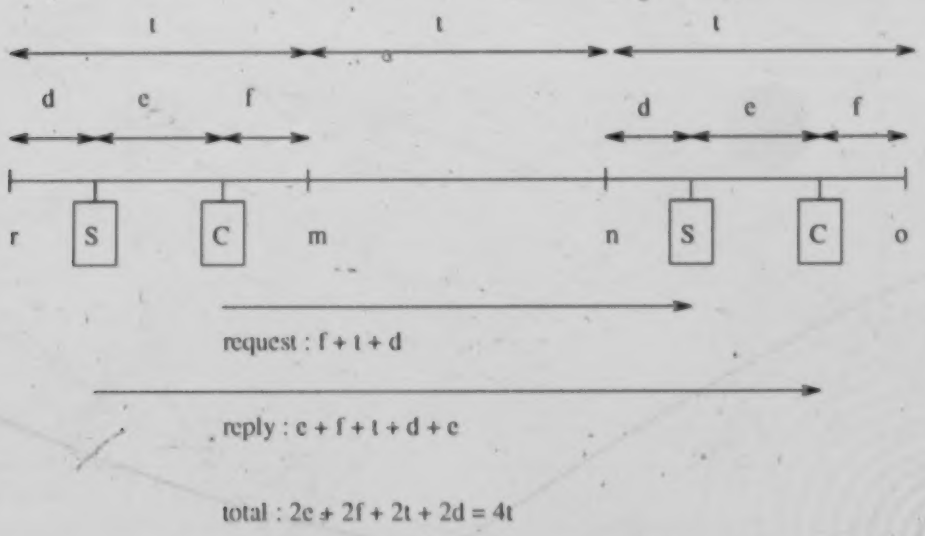
there are few servers, for example one, the entire traffic load is concentrated at the server. Thus, it is easy to see that, in such a case, it takes a longer time for a client to receive a reply. If there are more servers, the load is spread among the servers, and hence the result will be a lower response delay. For Ethernet, the result is different. Figure 6.19 shows that when there is only one server, Ethernet performs better than the cases when there are two, three or four servers. Starting at five servers, Ethernet performs better as the number of servers increases. If there is only one server in the network, the traffic is very concentrated. Thus, there are less collisions than the cases of two, three or four servers. If there are more servers, at about five, each server handles less of the entire network load. A client has to wait a shorter amount of time for a response, although there are more collisions in the bus.

6.3.3 Performance with Servers at Various Locations

The performance of Ethernet, Token-Bus and Expressnet is investigated with servers at various locations. Simulations are run with servers at the center and at the end of the bus. For Token-Bus, there is very slight difference. The model with the servers at the end of the bus has a higher response delay than the model with servers at the center. However, the difference is insignificant, and is not obvious to see. In Expressnet, the two models do not show much difference either. In fact, the total distance traveled by the request and the reply is independent of the locations of file servers. In fact, the total distance is a constant, as explained below.

The bus is stretched to a straight line: its total length is $3t$, as shown in Pictures 6.1 - 6.2, where t is the end-to-end propagation delay. There are two cases. In

Picture 6.1, server S is in the upstream of the bus. In Picture 6.2, client C is in the upstream of the bus. A transmission signal is propagated from r to o, where r is the head of the bus line and o is the end of the bus line; rm is the outbound portion of the bus. no is the inbound portion. r is at the most upstream and m is at the lowest point of the stream.

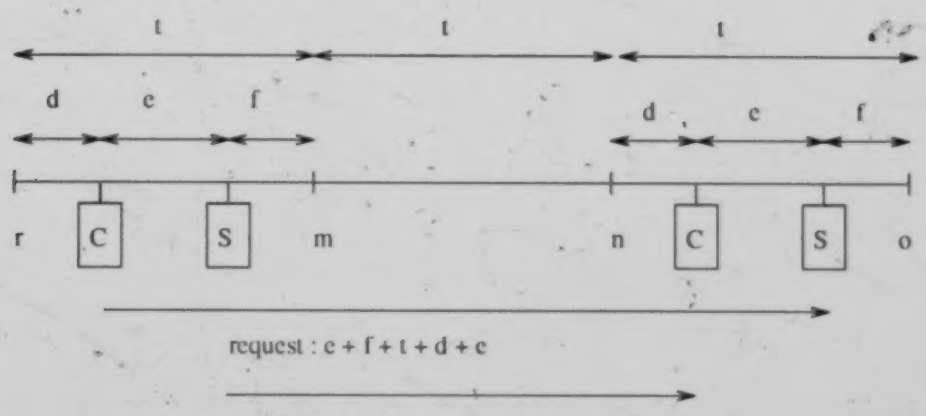


Picture 6.1

In case 1, C transmits a request which travels a distance of $f+t+d$ to reach S. Having received the request, server S transmits a reply back to C, and the distance traveled by the reply is $e+f+t+d+e$. The total distance traveled is $2e+2f+2t+2d$ which is $4t$.

In case 2, C transmits a request to S. The distance traveled by the request is $e+f+t+d+e$. S transmits a reply back to C. The distance traveled by the reply is $f+t+d$. The total distance traveled is again $2e+2f+2t+2d$ which is $4t$.

These two cases indicate that the total distance traveled by a request and a reply is always $4t$.



Picture 6.2

For Ethernet, there is a difference. The model with the server at the center has a lower response delay than the model with the server at the end of the bus link. The difference is distinguishable (Figure 6.23 - 6.26). When a server is located at the end of a bus link, the reply transmitted from it has a higher probability of colliding with other packets than the case when the server is located at the center, because in such a case other stations take a longer time to detect the signal of the transmission.

6.3.4 Comparison between Token-Bus and Token-Priority

Token-Priority has a better performance by having lower response delay and higher channel capacity than Token-Bus (Figure 6.3, Figure 6.11, Figure 6.15 - 6.18), especially when there are few servers. The performance of Token-Priority is even better than Ethernet which performs better than Token-Bus in the file server model when there are few servers. In a file server system, normally, there are long queues of reply messages at server stations. The work load of a file server station is especially heavy when there are few other servers. In Token-Priority, it takes less time for a reply message to wait for transmission since the server has the right to hold the token and clear the reply queue, when the number of requests at the queue exceeds or is equal to 2. As a result stations receive reply messages in a shorter time. Thus, more traffic can be generated and a higher channel capacity is obtained.

6.4 Conclusion

The simulations show that, unlike the uncorrelated traffic pattern, when there are few servers and more stations, Ethernet performs better than CSMA/CD-DP and Token-Bus. CSMA/CD-DP has the worst performance in any case. This is due to the long transmission delay in synchronized mode. Expressnet still has the best performance, since it has the shortest average round length. For Token-Bus, Expressnet, and CSMA/CD-DP, the network with more servers has better performance. For example, the average response delay of the network with 41 servers is less than that with 11 servers. But in Ethernet, the performance of the network with one server is better than the networks with two, three or four servers. The protocols

of Token-Bus and Expressnet is fair in the sense that the response delay does not show significant difference with servers at various locations. However locations of servers make difference in the case of Ethernet due to the fact that the packet from the server at the end of the bus link has a higher probability of colliding with other packets. The performance of Token-Bus can be improved by allowing the server to clear the reply queue when the number of reply messages waiting for transmission reaches a certain value.

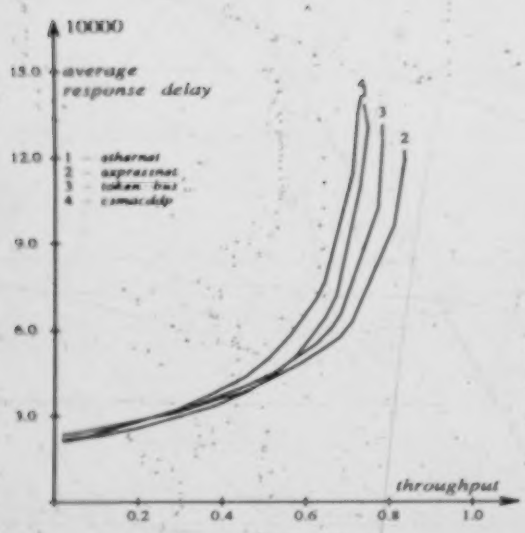


Figure 6.1 File server average response delay vs throughput, 11 stations, server = station 5

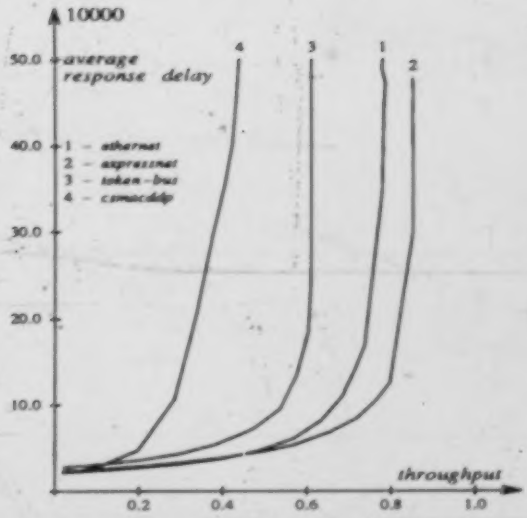


Figure 6.2 File server average response delay vs throughput, 41 stations, server = station 20

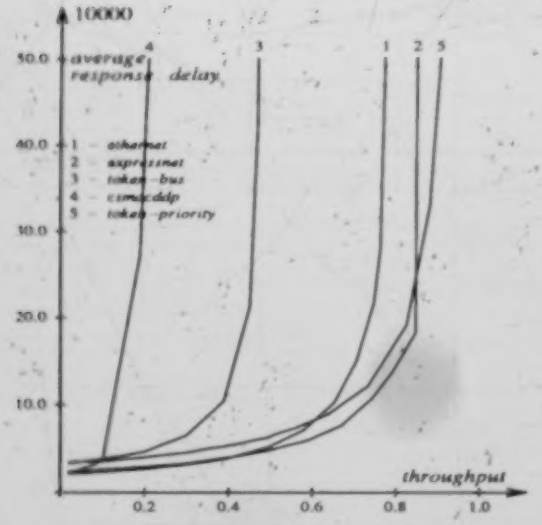


Figure 6.3 File server average response delay vs throughput, 81 stations, server = station 40

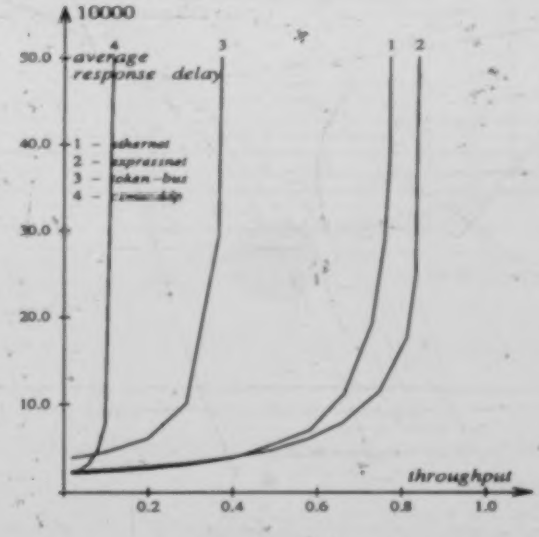


Figure 6.4 File server average response delay vs throughput, 121 stations, server = station 60

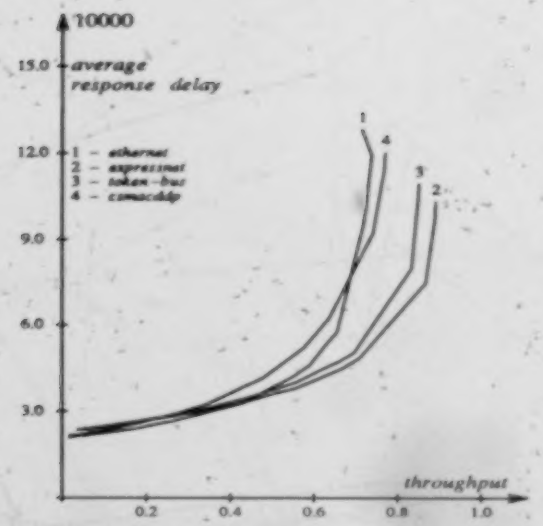


Figure 6.5 File server average response delay vs throughput, 11 stations, server = station 5, 6

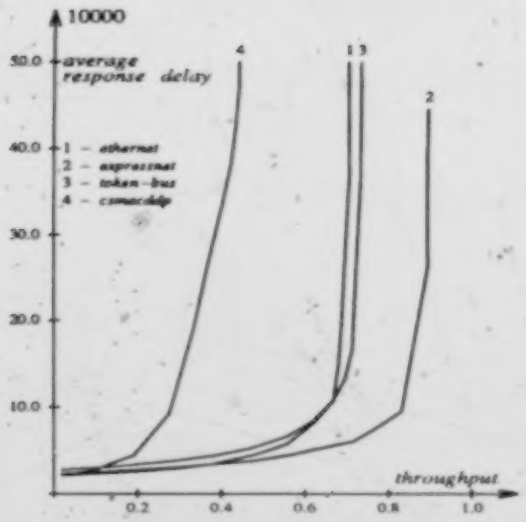


Figure 6.6 File server average response delay vs throughput, 41 stations, server = station 40, 41

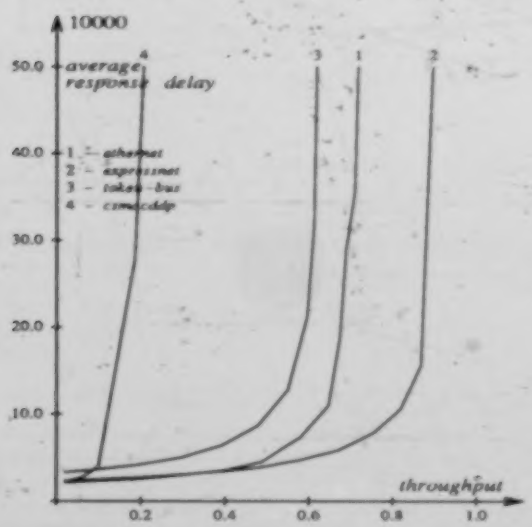


Figure 6.7 file server average response delay vs throughput, 81 stations, server = station 40, 41

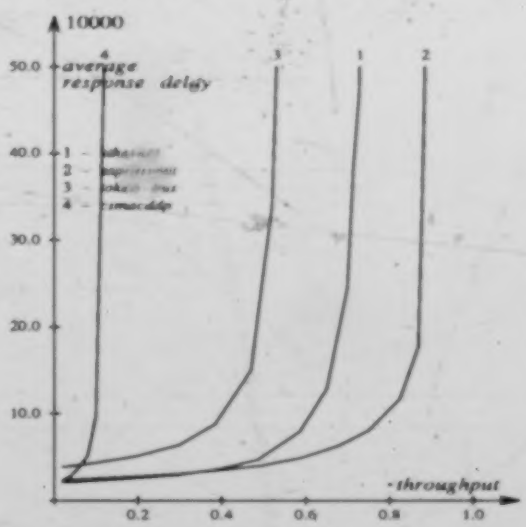


Figure 6.8 file server average response delay vs throughput, 121 stations, server = station 60, 61

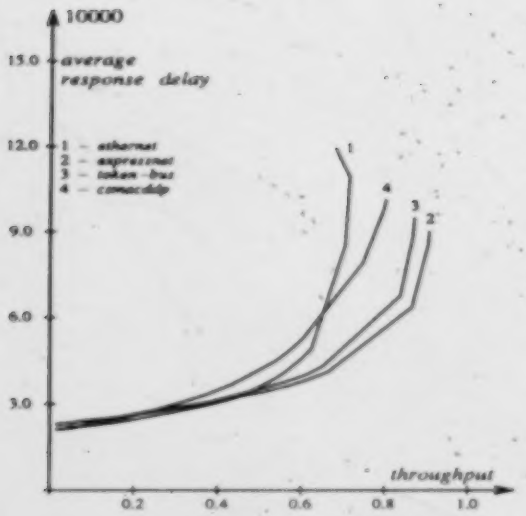


Figure 6.9 File server average response delay vs throughput, 11 stations, server = station 4 5 6

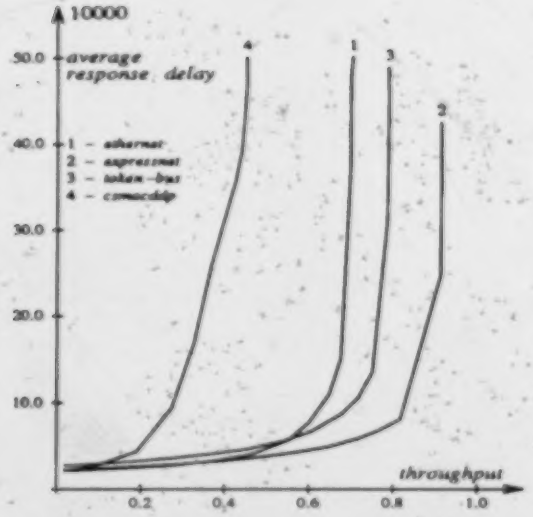


Figure 6.10 File server average response delay vs throughput, 41 stations, server = station 19 20 21

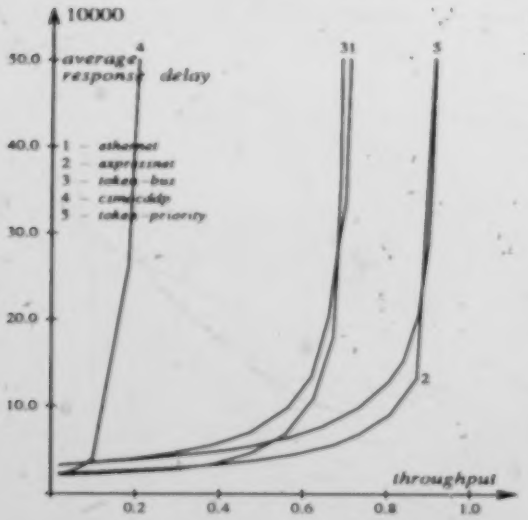


Figure 6.11 File server average response delay vs throughput, 81 stations, servers = station 40, 41,42

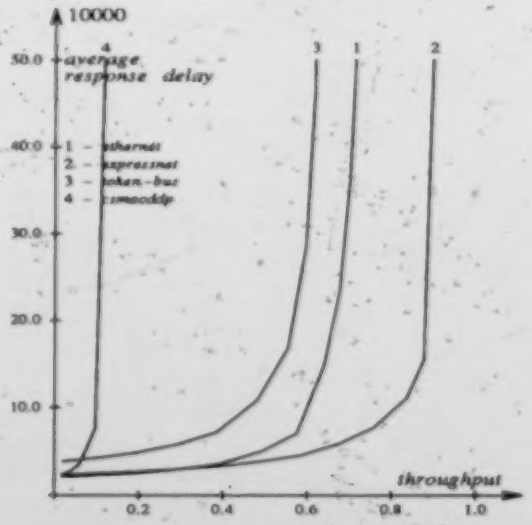


Figure 6.12 File server average response delay vs throughput, 121 stations, servers = station 59, 60, 61

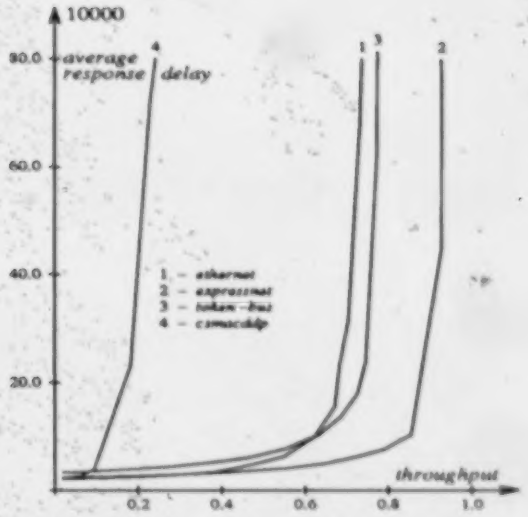


Figure 6.13 File server, 81 stations
5 servers at the center.

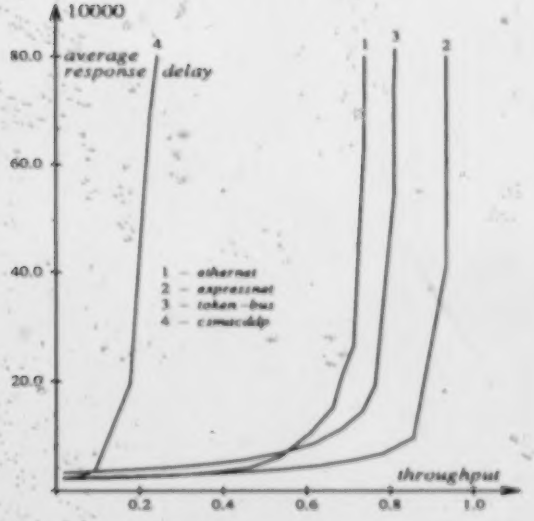


Figure 6.14 File server, 81 stations
7 servers at the center.

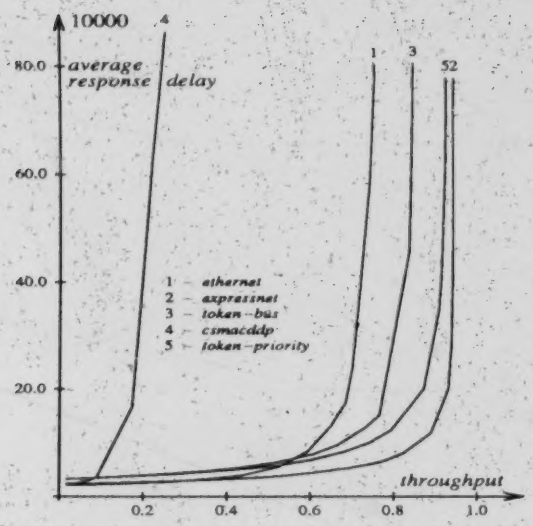


Figure 6.15 File server, 81 stations
10 servers at center.

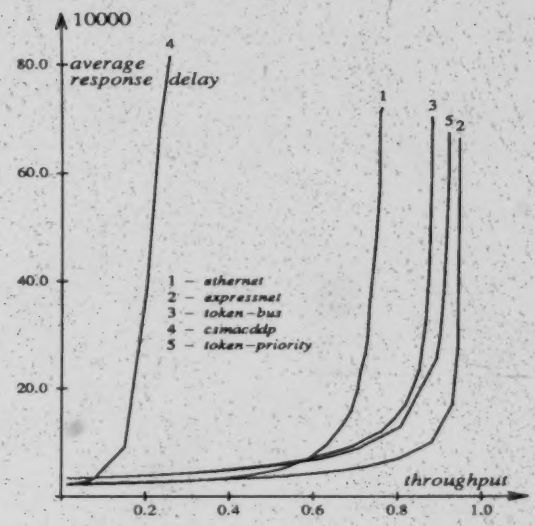


Figure 6.16 File server, 81 stations
20 servers at center.

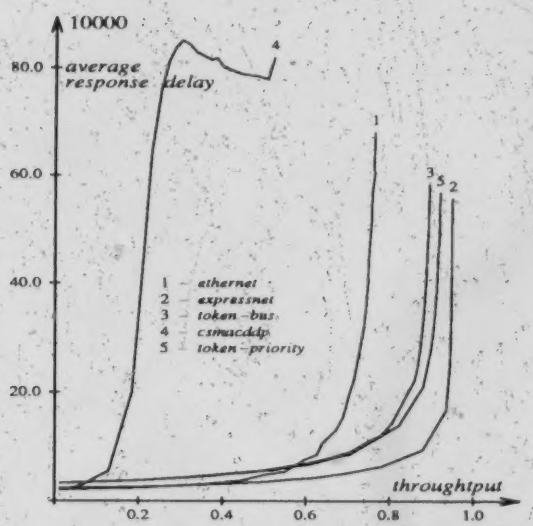


Figure 6.17 File server, 81 stations
30 servers at center.

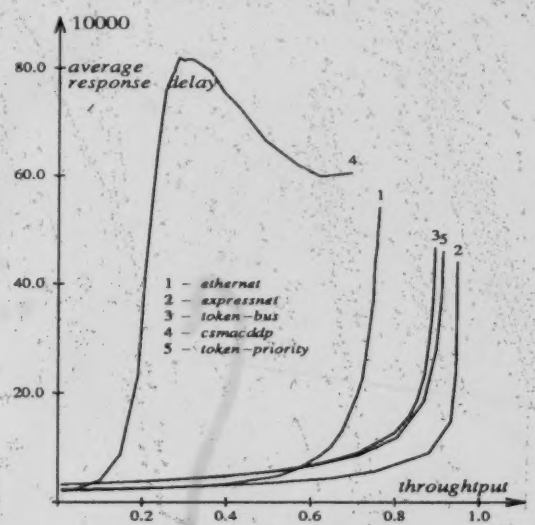


Figure 6.18 File server, 81 stations
40 servers at center.

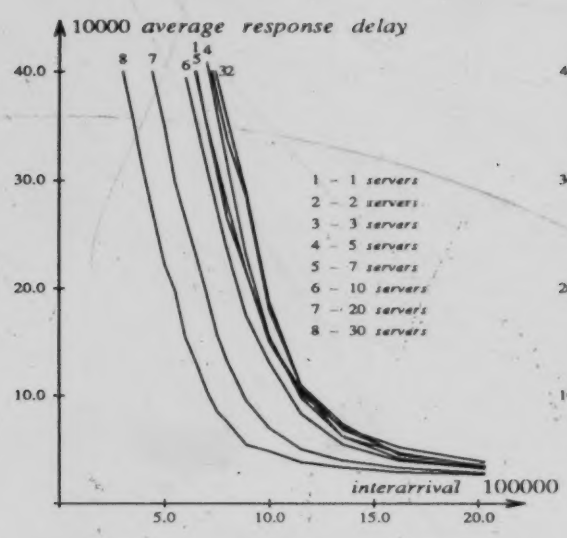


Figure 6.19 File server, Ethernet 81 stations, different number of servers

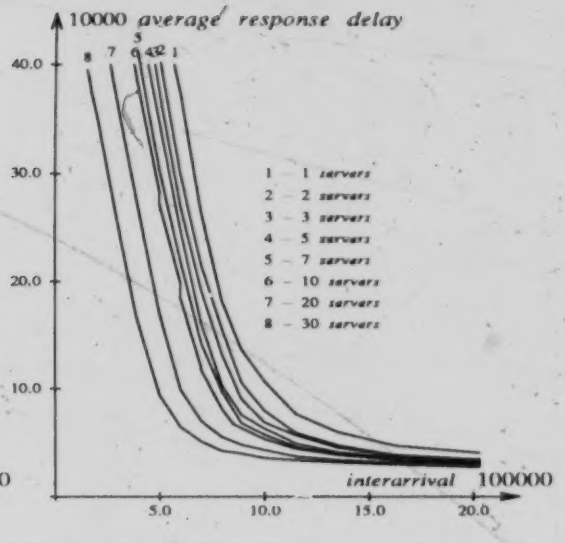


Figure 6.20 File server, Expressnet 81 stations, different number of servers

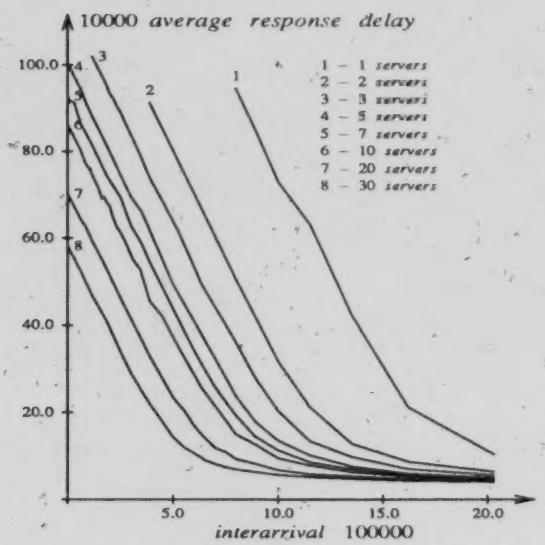


Figure 6.21 File server, Token-Bus 81 stations, different number of servers

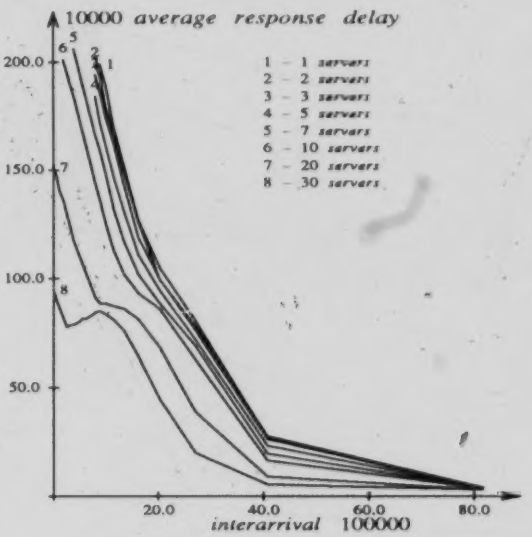


Figure 6.22 File server, CSMACDDP 81 stations, different number of servers

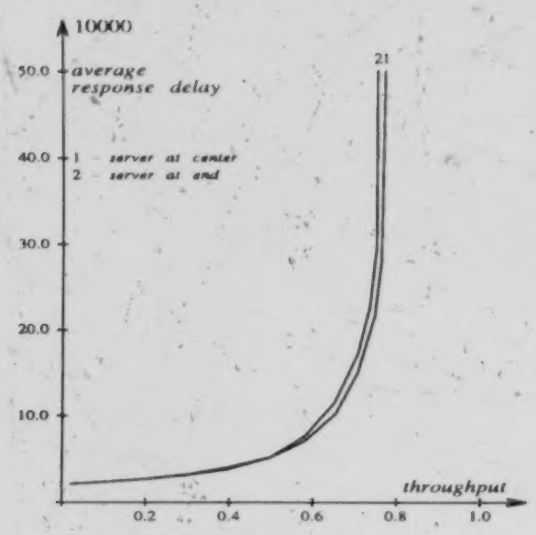


Figure 6.23 Ethernet, average response delay vs throughput, 81 stations, 1 server

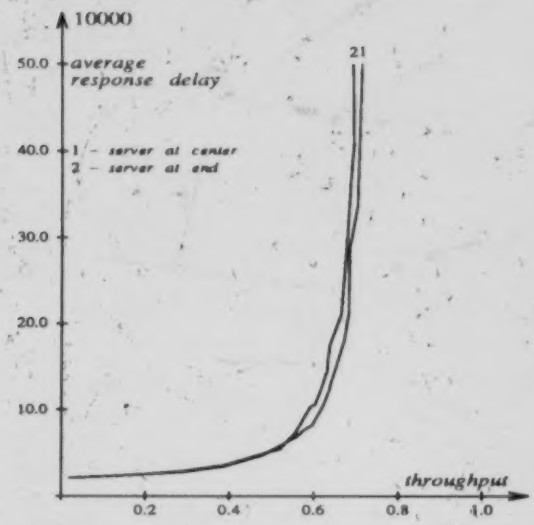


Figure 6.24 Ethernet average response delay vs throughput, 81 stations, 3 servers

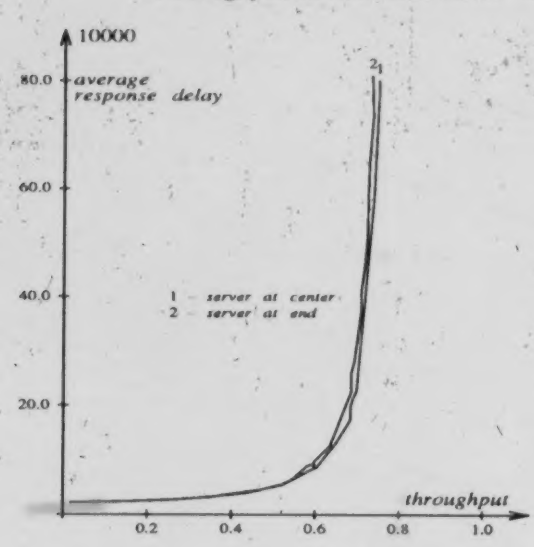


Figure 6.25 Ethernet average response delay vs throughput, 81 stations, 10 servers

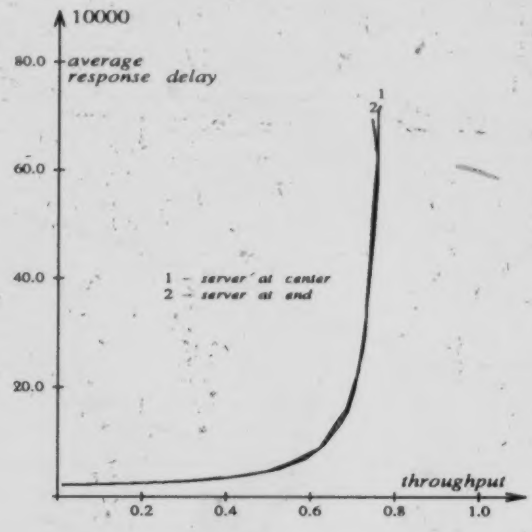


Figure 6.26 Ethernet average response delay vs throughput, 81 stations, 30 servers

Chapter 7

Summary

In this thesis, the performance of local area networks is investigated under uncorrelated traffic and two types of correlated traffic: systems with remote procedure calls (RPC traffic) and systems with file servers. Four different bus-type local area networks are selected for the investigation. They are Ethernet, Expressnet, Token-Bus and CSMA/CD-DP. A modified Token-Bus, Token-Priority, is also investigated to compare the performance of Token-Bus under different traffic patterns.

7.1 Performance with Uncorrelated Traffic

The results show that Ethernet has better performance over Token-Bus and CSMA/CD-DP when the traffic is light. This is due to the fact that Ethernet has the least overhead to transmit a packet. In Ethernet, when a station has a packet to transmit, it can transmit the packet whenever the bus is silent. The station has to do further actions only when a collision occurs. For Token-Bus and CSMA/CD-DP, a station has to wait for its turn to transmit even when the traffic is not heavy.

However, during moderate traffic, CSMA/CD-DP has the worst performance, due to the long transmission delay in synchronized mode. In this mode, according to the protocol, a station has to wait for a period of time before it starts to transmit a packet as explained in Section 2.1.4. When the traffic is extremely busy, CSMA/CD-DP performs even better than Token-Bus. This is explained by the fact that, in such a case, CSMA/CD-DP has a comparatively shorter average round length than that of Token-Bus.

Expressnet outperforms other local area networks, although Ethernet has better performance than Expressnet during very light traffic. This is because, as explained in Section 4.3.5, Expressnet has the shortest average round length. In Expressnet, if a station has a packet to transmit, it transmits a preamble when the bus is detected silent. During this period if there is no collision, the station will transmit its packet. The transmission overhead of Expressnet is much less as compared to those of Token-Bus and CSMA/CD-DP. The results also indicate that Token-Priority outperforms Token-Bus under busy traffic. In uncorrelated traffic, messages are generated in a station only according to a statistical distribution. It is very likely that a station may have a long queue of messages. In Token-Priority, a station can transmit more than one packet and clear the queue when it is holding the token. But for Token-Bus, a station can only transmit one packet per token. This difference makes Token-Priority a better protocol than Token-Bus.

7.2 Performance with RPC Traffic

In a remote procedure call system, after a caller process has transmitted a request message, it is very likely that it will suspend all its work just to wait for the reply message from the callee process. Traffic is generated according to this pattern. Performance of the local area networks are compared by looking at the average response delay. Two different cases are considered: stations with one process and stations with three processes. In general, the performance of the local area networks under RPC traffic does not differ very much from the case of uncorrelated traffic. The only notable point observed is that Token-Bus and Token-Priority have similar performance. This can be explained by the following: In an RPC system, request messages and reply messages are generated according to a traffic pattern. The intensity of the traffic depends on how long a request caller has to wait for the reply packet. Under such a traffic pattern, stations usually do not accumulate a long queue of messages. Thus it is not unexpected that Token-Priority and Token-Bus have similar performance. Ethernet has good performance during light traffic. CSMA/CD-DP has the highest response delay in moderate traffic, and performs better when the traffic is very busy. Such busy traffic will be generated when there are more processes in a station.

7.3 Performance with File Server Traffic

In a file server system, a client sends a message to a server, requesting file service. The client suspends all its work until it has received a reply. The traffic is generated according to this pattern. The performance analysis is made by measuring the

response delay and the throughput. Four cases are investigated.

Case 1: The performance of the local area networks is compared with same number of servers at the center. Result shows that if there are few servers and especially when the network has more stations, Ethernet has low response delay and higher channel capacity than Token-Bus and CSMA/CD-DP. This is explained by the fact that file server traffic is concentrated heavily at the file server. Such concentrated traffic favours Ethernet in the sense that the probability of collision is low. On the other hand, the concentrated traffic does not do any favours to Token-Bus and CSMA/CD-DP which have more control on the access to the bus link, and a server has to wait for its turn to transmit a reply. Expressnet has the best performance over other local area networks. This is because Expressnet has a shorter average round length as compared to other networks. A server waits a comparatively a short amount of time to transmit the next reply packet.

Case 2: This case intends to find out if a location of a server will affect the performance. In general, the performance of a file server system is not affected too much by the location of a server. The protocol of Expressnet and Token-Bus do not favor any specific location of a server. However, in Ethernet, the location of servers makes some difference. Ethernet has a higher response delay and lower channel capacity when the server is located at the end of the bus link. When the file server is at the end of the bus, it takes a longer time for other stations to detect its transmission, and collisions are more likely to occur.

Case 3: In this case, the performance of a local area network is compared with different number of servers. Results show that a network servers has shorter average response delay when there are more servers. The traffic load of a server is reduced if there are more servers. Thus a client will wait for a shorter time for the reply.

Case 4: This case is to compare the performance of Token-Bus with Token-Priority. Result shows that Token-Priority has a better performance than Token-Bus when there are few servers and large number of stations. In a file server system, it is very likely that a server will have a long queue of replies waiting for transmission. A station, in Token-Priority, can transmit more than one packet to clear the queue when it holds the token if the number of replies waiting for transmission is equal to or greater than a predefined value. Thus, a reply does not have to wait a long time to transmit.

7.4 Conclusion

The result of the thesis conclusively indicates that Ethernet, Expressnet and Token-Priority perform better than Token-Bus and CSMA/CD-DP under file server traffic when there are few servers. The performance of the networks under RPC traffic is similar to the performance under uncorrelated traffic except that Token-Priority performs better than Token-Bus in uncorrelated traffic. They both have almost the same performance under RPC traffic. Ethernet has good performance when the traffic is light to moderate. A real local area network is populated by more than just one kind of traffic. Typically, most local area networks have RPC traffic and file server traffic. Other kinds of traffic that can be found in local area networks are

electronic mail traffic or even voice traffic. Voice traffic and electronic mail traffic have a nature similar to RPC traffic and uncorrelated traffic respectively.

A good local area network should have good performance under a mixture of typical traffic patterns. CSMA/CD-DP has good performance only if the uncorrelated traffic and the RPC traffic is extremely heavy. In any other situation, CSMA/CD-DP has the worst performance. Token-Bus has good performance under heavy uncorrelated and heavy RPC traffic. In a file server system with few servers, Token-Bus has poor performance. As far as real traffic of a local area network is concerned, Figure 7.1- 7.5 indicates that Ethernet, Token-Priority and Expressnet provide more consistent and better performance than Token-Bus and CSMA/CD-DP.

The majority of traffic found in a local area network is file server traffic although there are no statistics found in publications. In most file server systems, usually there are only few servers (usually 1, 2 or 3) with a large number of clients. Moreover, in most local area networks the traffic is not busy. All the evidence shows that Ethernet is a good choice. There are many local area networks and file server systems installed in commercial companies, research institutes and universities. The results of this thesis supports the conclusion that Ethernet is a good choice.

Recently there has been an increasing interest in integrating voice and data in local area networks. Mixed traffic of this type is expected to be popular and common in the future. As a continuation of this thesis, it appears worthwhile to investigate the performance of various local area networks under this new type of traffic. In some other applications (such as CAD and image processing), it is sometimes necessary to transmit lengthy records of data. It would also be interesting

to see how routine algorithms and flow control algorithms perform in a wide area network under traffic containing long record transmissions.

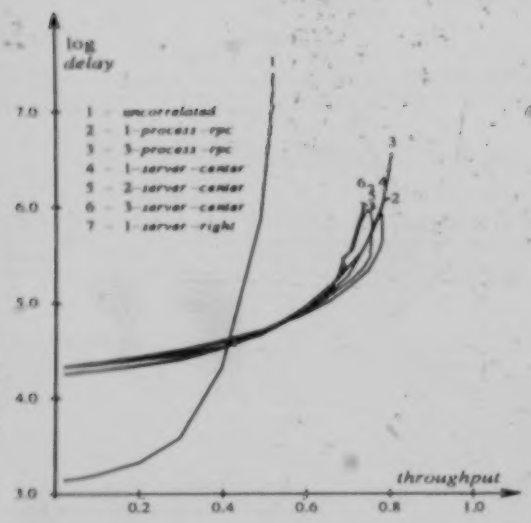


Figure 7.1, Ethernet performance with difference traffic, 81 stations, delay vs throughput

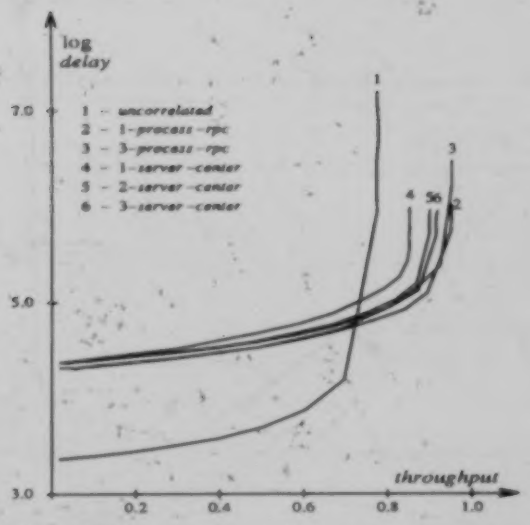


Figure 7.2, Expressnet performance with difference traffic, 81 stations, delay vs throughput

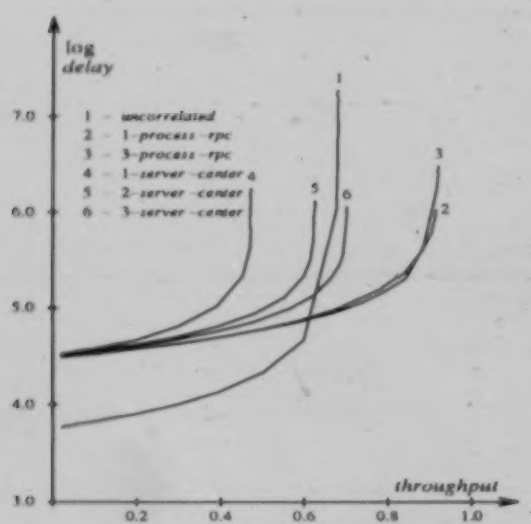


Figure 7.3, Token-Bus performance with differenc traffic, 81 stations, delay vs throughput

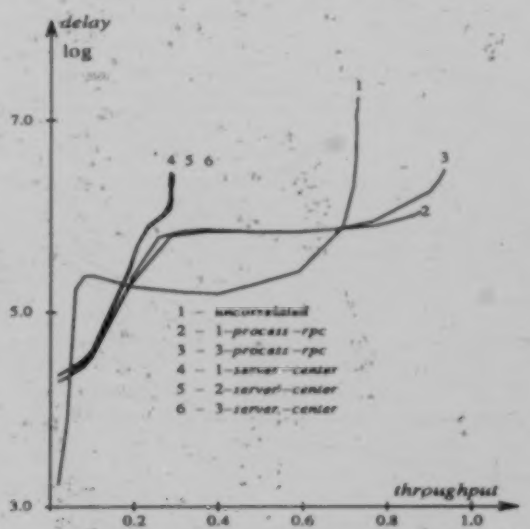


Figure 7.4, CSMACDDP performance with difference traffic, 81 stations, delay vs throughput

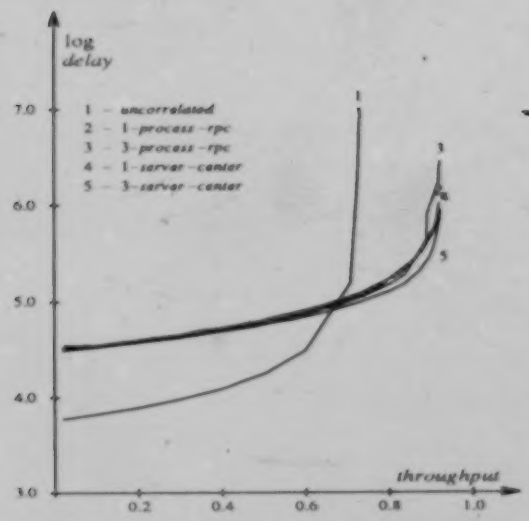


Figure 7.5, Token-Priority performance with difference traffic, 81 stations, delay vs throughput

Appendix A

Programs of Protocols

A.1 Ethernet

Two process :

- **transmitter()**: it transmits a packet upon the satisfaction of interpacket space, listening to the bus, aborting the transmission in the case of collision, transmitting the packet again after the back-off and ending a successful transmission.
- **receiver()**: it waits for the packet transmitted to it, and make sure that the packet is addressed to it.

```
transmitter () {

  TIME    last_silence, idle_period, backoff ();

  switch (the_action) {
```

```
case INITIALIZE:
case NEXT_PACKET:
  if (!get_packet (BUFFER, minpl, maxpl, frame)) {
    wait_event (CLIENT, MESSAGE_ARRIVAL, NEXT_PACKET);
    return;
  }
  the_station->collision_counter = 0;
case RETRY:
  if (undef (last_silence = last_eoa_sensed (BUS))) {
    wait_event (BUS, SILENCE, RETRY);
    return;
  }
  if ((idle_period = current_time - last_silence) < space) {
    wait_event (TIMER, space - idle_period, TRANSMIT);
    return;
  }
case TRANSMIT:
  transmit_packet (BUS, BUFFER, END_PACKET);
  wait_event (BUS, COLLISION, ABORT_PACKET);
  return;
case END_PACKET:
  stop_transfer (BUS);
  release_packet (BUFFER);
  continue_at (NEXT_PACKET);
```

```
    case ABORT_PACKET:
      abort_transfer (BUS);
      the_station->collision_counter ++;
      emit_jam (BUS, jam, STOP_JAM);
      return;
    case STOP_JAM:
      end_jam (BUS);
      wait_event (TIMER, backoff (), RETRY);
      return;
  }
}

receiver () {

  switch (the_action) {
    case INITIALIZE:
    case WAIT_FOR_PACKET:
      wait_event (BUS, END_MY_PACKET, PACKET_RECEIVED);
      return;
    case PACKET_RECEIVED:
      accept_packet (the_packet, the_port);
      wait_event (TIMER, 1, WAIT_FOR_PACKET);
      return;
  }
}
```

A.2 Token-Bus

Two processes :

- **transmitter()**: it transmits regular packets and passes the token to the next station.
- **receiver()**: it receives regular packets and the token.

```
transmitter  () {
  switch (the_action) {
    case INITIALIZE:
      the_station->token_used = NO;
      if (the_station->id == 0) {
        make_token ();
    case PASS_TOKEN:
        the_station->display_status = 'S';
        wait_event (TIMER, packet_space, SPACE_OBEYED);
        return;
      }
    case WAIT_FOR_TOKEN:
      the_station->display_status = 'W';
      wait_event (SIGNAL, TOKEN_ARRIVAL, GOT_TOKEN);
      return;
    case GOT_TOKEN:
      the_station->token_used = NO;
      continue_at (PASS_TOKEN);
```

```
    case SPACE_OBEYED:
      if (the_station->token_used || !get_packet (PACKET_BUFFER,
            min_packet_length, max_packet_length,
               frame_info_length))       {
        the_station->display_status = 'P';
        TOKEN_BUFFER->receiver = id_to_station
                                    (the_station->successor);
        transmit_packet (BUS, TOKEN_BUFFER, TOKEN_PASSED);
      } else {
        the_station->display_status = 'T';
        the_station->token_used = YES;
        transmit_packet (BUS, PACKET_BUFFER, PACKET_TRANSMITTED);
      }
      return;
    case TOKEN_PASSED:
      stop_transfer (BUS);
      continue_at (WAIT_FOR_TOKEN);
    case PACKET_TRANSMITTED:
      stop_transfer (BUS);
      release_packet (PACKET_BUFFER);
      continue_at (PASS_TOKEN);
  }
}

receiver () {
```

```
  switch (the_action) {
    case INITIALIZE:
    case WAIT_FOR_PACKET:
      wait_event (BUS, END_MY_PACKET, END_OF_PACKET);
      return;
    case END_OF_PACKET:
      if (the_packet->type == TOKEN_PACKET) {
        *TOKEN_BUFFER = *the_packet;
        internal_signal (TOKEN_ARRIVAL);
      } else
        accept_packet (the_packet, the_port);
      skip_and_continue_at (WAIT_FOR_PACKET);
  }
}
```

A.3 CSMA/CD-DP

Two Processes:

- **transmitter()**: it transmits packets, listens to the bus, shifts to synchronized mode when collisions occur, and returns to normal mode when the bus is quit for an amount of time equal to, $2n\tau$, where n is number of the stations τ is the end-to-end propagation delay.
- **receiver()**: it receivers packets.

```
transmitter () {
```

```
TIME  last_silence, idle_period;

switch (the_action) {
  case INITIALIZE:
    the_station->synchronized_mode = NO;
    the_station->active = NO;
  case MONITOR_BUS:
    wait_event (BUS, ACTIVITY, ACTIVITY_SENSED);
    wait_event (CLIENT, MESSAGE_ARRIVAL, NEXT_PACKET);
    return;
  case NEXT_PACKET:
    if (!packet_ready) continue_at (MONITOR_BUS);
  case RETRY:
    last_silence = last_eoa_sensed (BUS);
    if (undef (last_silence)) continue_at (ACTIVITY_SENSED);
        idle_period = minus (current_time, last_silence);
    if (les (idle_period, space_length)) {
        wait_event (TIMER, minus(space_length, idle_period),
                    TRANSMIT);
        wait_event (BUS, ACTIVITY, ACTIVITY_SENSED);
        return;
    }
  case TRANSMIT:
```

```
    the_station->active = YES;
    transmit_packet (BUS, BUFFER, EOT_SENSED);
    wait_event (BUS, MY_COLLISION, COLLISION_SENSED);
    return;
  case ACTIVITY_SENSED:
    wait_event (BUS, COLLISION, COLLISION_SENSED);
    wait_event (BUS, EOT, EOT_SENSED);
    return;
  case EOT_SENSED:
    update_priority;
    if (the_station->active) {
      stop_transfer (BUS);
      release_packet (BUFFER);
      the_station->active = NO;
    }
    if (!the_station->synchronized_mode)
        continue_at (NEXT_PACKET);
    wait_event (BUS, ACTIVITY, ACTIVITY_SENSED);
    wait_event (TIMER, priority_delay, MY_SLOT);
    return;
  case COLLISION_SENSED:
    assert(the_station->synchronized_mode == NO,
                        "Transmitter: illegal collision");
    the_station->synchronized_mode = YES;
```

```
    if (the_station->active) {
      abort_transfer (BUS);
      the_station->active = NO;
      emit_jam (BUS, jam_length, JAM_SENT);
    } else
      wait_event (TIMER, plus (jam_length, collision_delay),
                  COLLISION_GONE);
    return;
  case JAM_SENT:
    stop_jam (BUS);
    wait_event (TIMER, collision_delay, COLLISION_GONE);
    return;
  case COLLISION_GONE:
    wait_event (TIMER, priority_delay, MY_SLOT);
    wait_event (BUS, ACTIVITY, ACTIVITY_SENSED);
    return;
  case MY_SLOT:
    if (packet_ready) continue_at (TRANSMIT);
    wait_event (BUS, ACTIVITY, ACTIVITY_SENSED);
    wait_event (TIMER, minus (turnaround_delay, priority_delay),
                RESUME_NORMAL);
    return;
  case RESUME_NORMAL:
    the_station->synchronized_mode = NO;
```

```
        wait_event (TIMER, margin, MONITOR_BUS);
        return;
    }
}

receiver () {

    switch (the_action) {
      case INITIALIZE:
      case WAIT_FOR_PACKET:
        wait_event (BUS, END_MY_PACKET, PACKET_RECEIVED);
        return;
      case PACKET_RECEIVED:
        accept_packet (the_packet, the_port);
        continue_at (WAIT_FOR_PACKET);
        return;
    }
}
```

A.4 Expressnet

Four processes:

- **transmitter()**: it transmits packets and is accompanied by two other processes which monitor the station's ports to detect the EOC(OUT) and EOT-(IN), and passes the signals back to the transmitter.

- **receiver()**: it receive packets.
- **op_synchronizer**: it monitors the outbound channel to detect the EOC(OUT). When EOC(OUT) is detected, it passes the signal back to the transmitter.
- **ip_synchronizer**: it monitors the inbound channel to detect the EOT(IN). When EOT(IN) is detected, it passes the signal back to the transmitter.

```
transmitter () {
  switch (the_action) {
    case INITIALIZE:
      *PREAMBLE = make_packet(the_station,NONE,preamble_length);
      if (the_station->id == 0) continue_at (TRANSMIT_PACKET);
    case WAIT_SYNC:
      wait_event (SIGNAL, SYNC_EAC, CHECK_BUFFER);
      wait_event (SIGNAL, SYNC_ETR, TRANSMIT_PACKET);
      return;
    case CHECK_BUFFER:
      if (! packet_pending) continue_at (WAIT_SYNC);
    case TRANSMIT_PACKET:
      transmit_packet (OP, PREAMBLE, START_TRANSFER);
      wait_event (OP, MY_COLLISION, PREEMPTED);
      return;
    case PREEMPTED:
      abort_transfer (OP);
```

```
      continue_at (WAIT_SYNC);
    case START_TRANSFER:
      if (packet_pending) {
        abort_transfer (OP);
        transmit_packet (OP, BUFFER, END_PACKET);
        return;
      } else {
        stop_transfer (OP);
        continue_at (WAIT_SYNC);
      }
    case END_PACKET:
      stop_transfer (OP);
      release_packet (BUFFER);
      continue_at (WAIT_SYNC);
  }
}

ip_synchronizer () {
  switch (the_action) {
    case INITIALIZE:
    case WAIT_FOR_EAC:
      wait_event (IP, EOT, WAIT_DT);
      return;
    case WAIT_DT:
      wait_event (TIMER, delta_t, END_TRAIN);
```

```
      wait_event (IP, ACTIVITY, NOT_END_TRAIN);
      return;
    case END_TRAIN:
      internal_signal (SYNC_ETR);
    case NOT_END_TRAIN:
      skip_and_continue_at (WAIT_FOR_EAC);
  }
}

op_synchronizer () {
  switch (the_action) {
    case INITIALIZE:
    case WAIT_FOR_EAC:
      wait_event (OP, EOT, CHECK_SYNC);
      return;
    case CHECK_SYNC:
      if (the_packet->sender != the_station)
        internal_signal (SYNC_EAC);
      skip_and_continue_at (WAIT_FOR_EAC);
  }
}

receiver () {

  switch (the_action) {
    case INITIALIZE:
```

```
    case WAIT_FOR_PACKET:
      wait_event (BUS, END_MY_PACKET, PACKET_RECEIVED);
      return;
    case PACKET_RECEIVED:
      accept_packet (the_packet, the_port);
      continue_at (WAIT_FOR_PACKET);
      return;
  }
}
```

Bibliography

[1] B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy, *Lightweight remote procedure call*, In Twelfth ACM Symposium on Operating Systems Principles, pages 102-103, 1989.

[2] A. D. Birrell and B. Nelson, *Implementing remote procedure calls*, ACM Transactions on Computer Systems, pages 39-59, Feb. 1984.

[3] J. Dion, *The Cambridge File Server*, Operating Systems Review, pages 26-35, Oct. 1980.

[4] L. Fratta, F. Borgonovo and F. Tobagi, *The Expressnet: a Local Area Communication Network Integrating Voice and Data*, In Proceedings of International Conference on Performance of Data Communication System and their Applications, Paris, France, Sept. 1981.

[5] P. Gburzynski, and P. Rudnicki, *The Lansf, Protocol Modeling Environment*, University of Alberta, Technical Report TR 89-19, Sept. 1989.

[6] J. L. Hammond, and R. O'Relly, *Performance Analysis of Local Computer Networks*, Reading, MA: Addison-Wesley, 1986.

[7] J.H. Howard, M.L.Kazar, S.G.Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M.J. West, *Scale and Performance in a Distributed File System*, ACM Transactions on Computer Systems, pages 51-81, Feb. 1988.

[8] W. Kiesel and P. Kuehn, *A New CSMA/CD Protocol for Local Area Networks with Dynamic Priorities and Low Collision Probability*, IEEE Journal on Selected Areas in Communications, pages 869-876, 1983.

[9] L. Kleinrock, and F. Tobagi, *Packet Switching in Radio Channels: Part I-Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics*, IEEE Transactions on Communications, pages 1400-1416, Nov. 1975.

[10] J. G. Mitchell, and J. Dion, *A Comparison of Two Network-Based File Servers*, Communications of the ACM, pages 333-245, Apr. 1982.

[11] S.J. Mullender and A. S. Tanenbaum. *Immediate files*, Software Practice and Experience, pages 365-368, Apr. 1984.

[12] J.K. Ousterhout, H. Da Costa, D. Harrison. J. A. Kunze, M. Kupfer, and J. G. Thompson, *A Trace-Driven Analysis of the Unix 4.2 BSD File System*, In The Tenth ACM Symposium on Operating Systems Principles, pages 15-24, Dec. 1985.

[13] M.D. Schroeder and M. Burrows, *Performance of Firefly RPC*, In Twelfth ACM Symposium on Operating Systems Principles, pages 83-90, 1989.

[14] W. Stallings, *A tutorial on the IEEE 802 Local Network Standard*, In R. Pickholtz, editor, Local Area & Multiple Access Networks, pages 1-30, Computer Science Press, 1986.

[15] W. Stallings, *Handbook of Computer Communications Standards*, Volume 2, Chapter 2 & 4, A Stallings/Macmillan Book, 1987.

[16] W. Stallings, *Local Network Performance.*, IEEE Communication Magazine, Feb. 1984.

[17] B. Stuck, and E. Arthurs, *A Computer Communication Network Performance Analysis Primer*, Englewood Coiffs, NJ: Prentice-Hall, 1985.

[18] H. Sturgis, J. Mitchell, and J. Israel, *Issues in the Design and Use of a Distributed File System*, ACM SIGOPS Operating System Review, pages 55-69, July 1980.

[19] L. Svobodova, *File Servers for Network-Based Distributed Systems*, Computing Surveys, Vol.16, No.4, Dec. 1984

[20] F. Tobagi, F. Borgonovo, and L. Fratta, *Expressnet: A High-Performance Integrated-Services Local Area Network*, IEEE Journal on Selected Areas In Communications, pages 898 - 913, Nov. 1983.

[21] F. Tobagi and V. B. Hunt, *Performance analysis of carrier sense multiple access with collision detection*, Computer Network Vol. 4, pages 245-259, 1980.